Cleaning, Climbing, Crawling Robots

SERVO MAGAZINE

FOR THE ROBOT INNOVATOR

www.servomagazine.com

December 2010

Building The ARDUINO BOT

You've met the brain, *Now* build the base!

Combat Zone

Calculating bolt torque

Keep your bot together while taking the hits.

Propelled By The Propeller

What can this Parallax chip do for your next build?

VEX Motor Control Experiments

Use simple VEX hardware to aim a solar panel and track the sun.

U.S. $5.50 CANADA $7.00

12

0 71486 02422 4

Vol. 8 No. 12

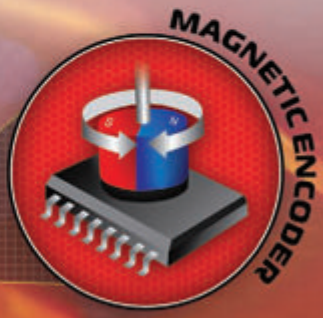
SERVO MAGAZINE

December 2010

INVEST in the BEST

YOUR ROBOT WILL THANK YOU

At Hitec, we know how much time and money you dedicate to your robot hobby. So why not make sure your servos are delivering what they promise? Our technologically advanced mega servos are tough enough for even your most challenging projects. Designed with our newest high resolution "G2.5" 12-bit programmable digital circuit and indestructible titanium gears, the HS-7980TH and HS-M7990TH give mega torque and speed with pinpoint accuracy. Built to last, these servos bring unprecedented power and sustainability to your investment.



Our HS-7980TH employs durable mechanical potentiometer technology while the HS-M7990TH utilizes the first ever magnetic encoder.

	6 Volts		7.4 Volts				
Model	Speed	Torque	Speed	Torque	Part#	Dimensions	Weight
HS-7980TH	0.20	500 oz.in	0.17	611 oz.in	37980S	1.72 x 0.88 x 1.57 in	2.70 oz
HS-M7990TH	0.20	500 oz.in	0.17	611 oz.in	37990S	1.72 x 0.88 x 1.57 in	2.70 oz

HITEC

We Serve you Right!

12115 Paine Street, Poway, CA 92064 • 858-748-6948 • www.hitecrcd.com

Thanks to Lynxmotion for the Phoenix robot featured in this advertisement.

X-TREME geek

x-treme Geek is Wired in to What You Want.

You want that Big Reaction when your loved one opens their gift. We guarantee you'll get it when you shop with us. From never before seen modern marvels to kitschy cool blasts from the past, X-treme Geek has the gifts to make this holiday the best one yet. Shop x-tremegeek.com.

Computer not included

2500941.....$16.95
EchoBot Voice Messenger

- Detects movement up to 3 feet away

EchoBot Delivers a Voice Message when Somebody Moves Nearby

Record messages for your friends and coworkers, and the EchoBot will play them back when they approach. Up to ten seconds of audio will play back when someone triggers the motion detector (built into the "eye," of course). Bendable legs and suction cup feet make it easy to position the EchoBot anywhere around your home, office or car. Colors vary. Size: 7" tall.

A Carpentry Kit for a Robot?

NEW

Maybe early robots were made of wood....or maybe it's just an awesome idea who's time is long overdue. This kit is supplied with pre-punched wooden boards, gears, shafts, switch, motor, battery holder, and all necessary parts to build your own wooden robot, complete with blinking eyes. Includes easy-to-follow instructions. Requires screwdriver and long nose pliers for construction, plus two AA batteries - all not included. Recommended for ages 10 and up.

3152147.....$19.95
Robomech Wooden Kit

⚠ **WARNING:**
CHOKING HAZARD—Small Parts.
Not for children under 3 yrs.

POW Robot T-Shirt

NEW

From R2-D2 and C3-PO to Johnny 5 and Rosie, there are few things cooler than robots. Now you can proudly sport your robot fandom in this cotton tee. Accented by comic book style POW background, the robot pictured on this shirt will WOW your friends while you look cool, calm, and comfortable. Made in America and printed with environmentally friendly inks. Color: Heather

POW Robot $24.95

3200150.....Medium
3200151.....Large
3200152.....X-Large
3200153.....XX-Large

Virtual Guard Dog Protects Your Home 24/7

Dog not included

- Adjustable volume & sensitivity
- 5¼"L x 5¾"W x 7½"H; 6' cord

Switch from barking to tranquil rainforest sounds for a soothing audio backdrop for a party!

1320672.....$89.95
Rex Plus, The Electronic Watchdog

Few security systems are as intimidating as an angry barking dog. The Rex Plus security system "sees" through walls and doors to detect when someone approaches your home and then it "barks" just like a real German Shepherd. The startlingly realistic barking increases in intensity as the interloper gets closer. You get inexpensive, reliable, 24-hour protection that's perfect for extended absences.

3200103.....$89.95
The Wizard's Wand Universal Remote

Cast A Magic Spell Over Your Electronics!

To truly rule the room and couch kingdom you must have a wizard on your side...or better yet, be a wizard yourself. With this vibrating wand as universal remote, controlling your home entertainment systems is simple sorcery. A flick of the wrist and a spin of your magical wand changes the channel, volume, track, and more, including rewind and fast forward. A total of 13 programmable commands are controlled by circular movements, up and down gestures, or back and forth whisps of this vibrating wand. It looks like magic, but it's really technology — the same accelerometer technology used in Wii remotes. With practice you'll master a level of wizard like skill to impress all your friends. The remote is recognized by almost every piece of modern home entertainment apparatus. Size: 39cm x 6.5cm x 3.8cm. Weight: 295g.

800.480.4335 • x-tremegeek.com

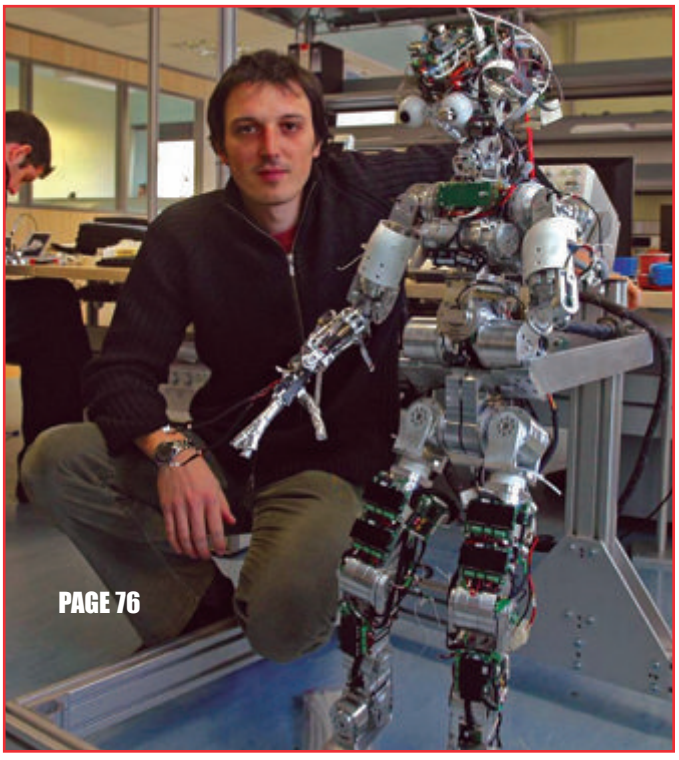
SERVO MAGAZINE

12.2010

VOL. 8 NO. 12

Did you know that each article in *SERVO Magazine* has its own webpage? It's where you go for downloads, comments, updates, corrections, or to link to the article in the digital issue. The unique link for each webpage is included with the article.

You can also visit article pages from back issues at **www.servomagazine.com**. Just select the ***Past Issues*** tab from the ***About SERVO*** drop down menu, click the Table of Contents link, and the article name.



Columns

The Combat Zone...

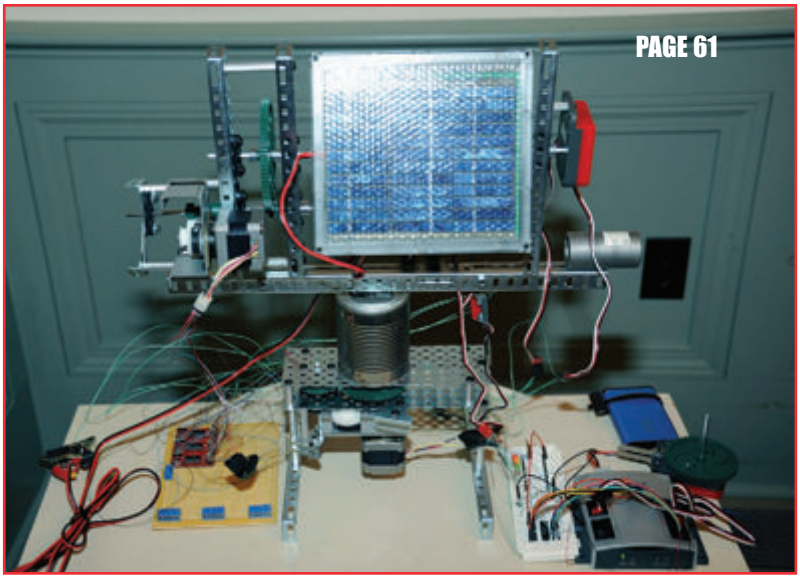
Features

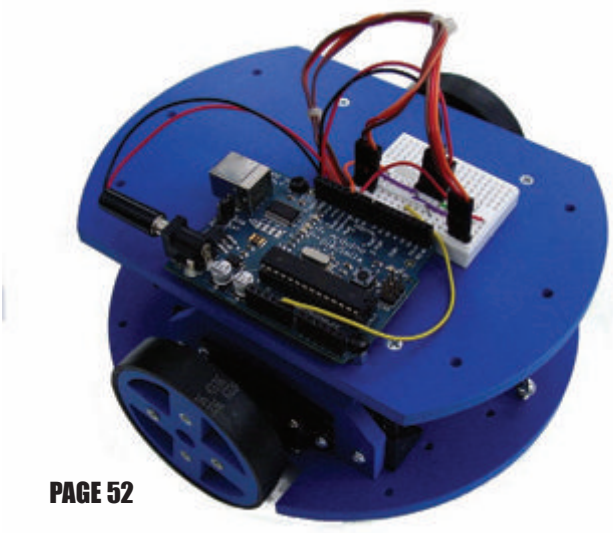
Events

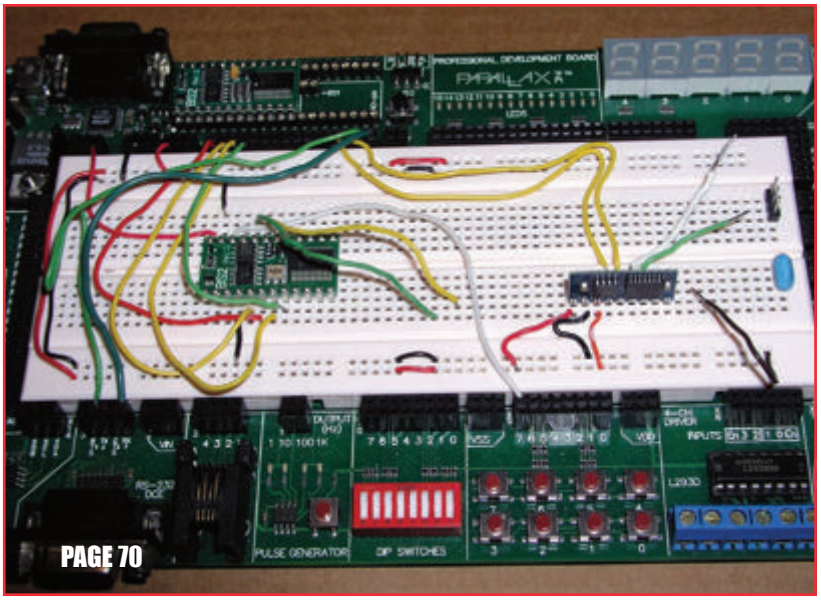
Departments

SERVO Magazine **(ISSN 1546-0592/CDN Pub Agree#40702530)** is published monthly for $24.95 per year by T & L Publications, Inc., 430 Princeland Court, Corona, CA 92879. PERIODICALS POSTAGE PAID AT CORONA, CA AND AT ADDITIONAL ENTRY MAILING OFFICES. POSTMASTER: Send address changes to **SERVO Magazine, P.O. Box 15277, North Hollywood, CA 91615** or Station A, P.O. Box 54, Windsor ON N9A 6J5; cpcreturns@servomagazine.com

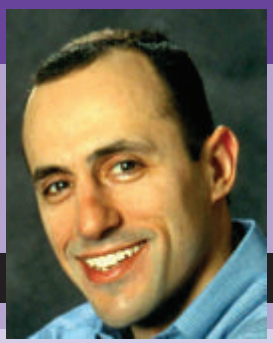
In This Issue ...







Mind / Iron



by Bryan Bergeron, Editor

Robot Control Options: Don't Forget Your Feet

When it comes to controlling a robot arm with six or more degrees of freedom or directing a rover through a complex maneuver, it can be challenging to direct movement in real time. There's only so much you can do with two hands and a standard game controller, and a cumbersome keyboard is often less than optimal.

There are a variety of commercial solutions, such as 3D joysticks. However, these tend to be expensive and optimized for software control, and not for directing a robot. Voice recognition is another option, if you use a dedicated voice recognition chip with low computational overhead.

Another option is to look outside of the traditional controller market, especially in the market that serves electronic musicians. For example, take a look at the range of music controllers available for piano, percussion, guitar, and other musical instruments. My favorite controllers in the music realm are the various foot pedals that enable guitarists to change their tone or volume without interrupting their guitar play. The simplest volume control foot pedals have a potentiometer that's coupled to the pedal. Press your toes down and the resistance goes up; press your heel down, and the resistance goes down.

The problem with most guitar volume control pedals is that they stay where you leave them. If you need a spring return like a gas pedal, then you can use a high-hat controller. These pedals are designed to operate virtual high-hat cymbals in an electronic drum set. You can see a profile of my pedal in **Figure 1**. This pedal happens to be made by Roland which is one of several manufacturers in this space.

You can also use music control pedals for on-off or single-shot applications. For example, take the piezo-based sensor that's used in place of a base drum. **Figure 2** shows my drum kicker attached to one of these piezo sensors. Stepping on the pedal causes the hammer to hit the piezo sensor which generates an impulse that can be sensed by a chip. Of course, you don't have to use a drum kicker to set off the sensor. As with the first foot controller, this sensor is made by Roland (**www.roland.com**), but equivalent units are available from Yamaha and others.

Figure 1

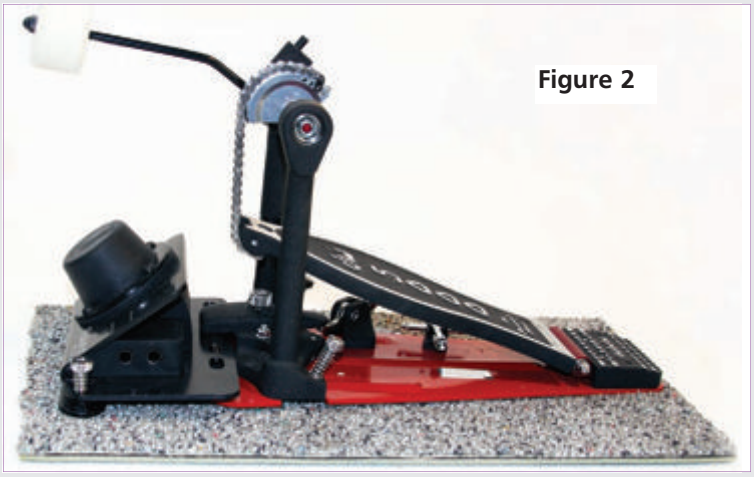


Figure 2

Other foot-operated sensors or switches include on-off switches sold for dictation machines and line-operated hand tools. My favorite in the latter category is the Proxxon FS foot controller (**www.proxxon.com**). It's good for about 5A at 120 VAC. For most robot control circuits, you'll want to use the switch to do something other than switch the mains connection to a power supply. You'll get more immediate response by switching the DC out of a supply because the filter capacitors retain a charge after the AC is disconnected.

The point of this discussion is that there are lots of off-the-shelf alternatives to game pads and joysticks out there. You just need to know where to look. I suggest that you use one of the music supply websites – such as **www.sweetwater.com** or **www.musiansfriend.com** – to learn the range of control options available. Then go to eBay and purchase what you need at a discount.

FOR THE ROBOT INNOVATOR SERVO MAGAZINE

Published Monthly By
T & L Publications, Inc.
430 Princeland Ct., Corona, CA 92879-1300
(951) 371-8497
FAX **(951) 371-3052**
Webstore Only **1-800-783-4624**
www.servomagazine.com

Subscriptions
Toll Free **1-877-525-2539**
Outside US **1-818-487-4545**
P.O. Box 15277, N. Hollywood, CA 91615

PUBLISHER
Larry Lemieux
publisher@servomagazine.com

ASSOCIATE PUBLISHER/ VP OF SALES/MARKETING
Robin Lemieux
display@servomagazine.com

EDITOR
Bryan Bergeron
techedit-servo@yahoo.com

CONTRIBUTING EDITORS
Jeff Eckert, Jenn Eckert, Tom Carroll, David Geer, Dennis Clark, R. Steven Rainwater, Fred Eady, Kevin Berry, Daniel Ramirez, John Blankenship, Gregory Intermaggio, Samuel Mishal, Gordon McComb, Thomas Kenney, Matthew Spurk, Katherine Kelly, Pete Smith

CIRCULATION DIRECTOR
Tracy Kerley
subscribe@servomagazine.com

MARKETING COORDINATOR WEBSTORE
Brian Kirkpatrick
sales@servomagazine.com

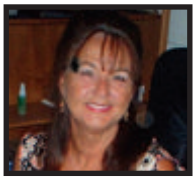
WEB CONTENT
Michael Kaudze
website@servomagazine.com

ADMINISTRATIVE ASSISTANT
Debbie Stauffacher

PRODUCTION/GRAPHICS
Shannon Christensen

If all else fails, you can resort to deconstructing Wii remotes and other devices. However, it's probably better to spend your time and efforts on the robotics, and let someone else engineer the controllers. That is, unless your focus is robotic control and communications. If this fits your profile, then you've probably been trying to figure out how to use one of those mind-control interfaces such as the one sold with Mattel's MindFlex Game (**www.mindflexgames.com**) to control your robot.

Mind control may be the ultimate robot controller, but we're not there yet. So that means you've got an opportunity to make it happen ... what are you waiting for? **SV**

Robytes

by Jeff and Jenn Eckert

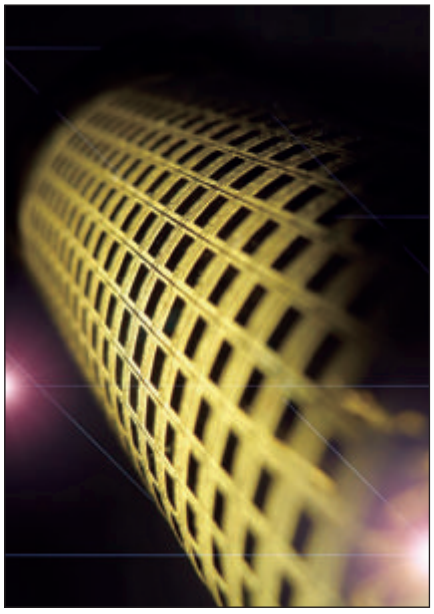
Bots Protecting Nuke Site

MDARS robots now guard the Nevada National Security Site.

Back in 2000, Congress established the National Nuclear Security Administration (NNSA, **nnsa.energy.gov**) to look after the nation's stockpile of nuclear weapons, nuclear proliferation programs, and issues relating to the US Navy's nuke propulsion systems. Some 64 miles outside of Las Vegas is NNSA's Nevada National Security Site — a 1,360 square mile facility that played host to more than 1,000 nuclear weapons tests and now houses tens of millions of cubic feet of low-level radioactive waste. In a move designed both to enhance security and save $6 million by avoiding infrastructure investments for cameras, lights, cables, and trenches, etc., the NNSA has deployed a team of three mobile detection assessment response system (MDARS) robots to take over security duties from humans. The bots — under development by General Dynamics Robotic Systems (**www.gdrs.com**) since 1993 — sport onboard sensors and video to allow a remote operator to see intruders and suspicious activity. However, the MDARS operate independently, so the operator just hangs out until one of them requires human assistance in assessing a situation. The diesel-powered vehicles can zip along at 20 mph (32 kph), operate for about 16 hours without refueling, and detect intruders at a distance of about 650 ft (200 m). They can even monitor RFID inventory. Reportedly, they have been tested with automatic weapons at their disposal, but there is no indication that these units will be armed. Either way, you probably don't want to mess with them.

New Take on E-Skin

The problem of equipping bots with tactile capabilities is an old one, and various attempts to develop artificial skin have met with limited success. According to a U.C. Berkeley (**www.berkeley.edu**) research team however, this is because previous efforts have relied on organic materials which are flexible and relatively easy to process. According to Ali Javey, head of the team and also a scientist at the Lawrence Berkeley National Lab, "The problem is that organic materials are poor semiconductors which means electronic devices made out of them would often require high voltages to operate the circuitry." Javey's folks have taken a different approach by creating a touch-sensitive artificial skin out of nanowires. Basically, they grow germanium/silicon nanowires on a cylindrical drum and then roll it over a sticky substrate, thereby depositing them in an orderly fashion. Their version is printed on polyimide film, but they say the technique works on other plastics, paper, glass, and other substances. The bottom line is a skin that requires less than 5V to operate and can detect pressures from 0 to 15 kilopascals (about 2.2 PSI). It has been successfully tested to more than 2,000 bending cycles. Look for it someday on a bot that's handing you an egg, pouring a glass of wine, or giving you a Swedish massage.



A new nanowire-based artificial skin developed at U.C. Berkeley.

Programmed for Deception

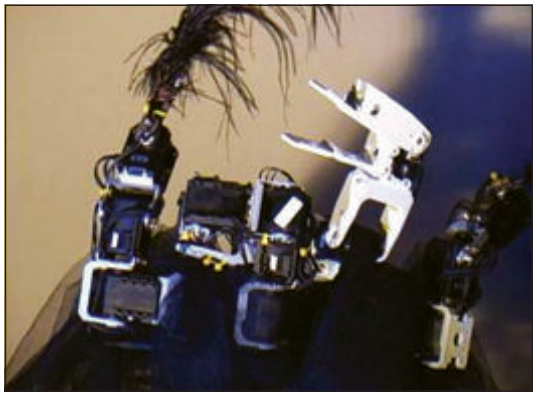
Asimov's Three Laws address bots' rules of behavior in terms of preserving self and human life, but they leave open the option of lying, cheating, deception, and various forms of skullduggery. In what is believed to be the first detailed examination of robot deception, Prof. Ronald Arkin and engineer Alan Wagner, working out of Georgia Tech (**www.gatech.edu**), have "developed algorithms that allow a robot to determine whether it should deceive a human or other intelligent machine, and we have designed techniques that help the robot select the best deceptive strategy to reduce its chance of being discovered." The relatively simple study focused on "actions, beliefs, and communications of a

Georgia Tech's Ronald Arkin and Alan Wagner watch as one bot fakes out another. Photo by Gary Meek.

robot attempting to hide from another robot to develop programs that successfully produced deceptive behavior." Specifically, in a game of hide and seek, the deceiving bot knocked down markers in a way that would indicate that it would be hiding in one of three locations, then actually hid in one of the other two. After a run of 20 experiments, the hider deceived the seeker 75 percent of the time.

The concept clearly has its uses in applications such as military search and rescue (and, in fact, the investigation was funded by the Office of Naval Research); misleading the enemy is a useful capability. But, Arkin noted, "We have been concerned from the very beginning with the ethical implications related to the creation of robots capable of deception, and we understand that there are beneficial and deleterious aspects." Indeed, a world crawling with robotic lawyers, car salesmen, investment counselors, and politicians is the stuff of nightmares.

Roll Over, Tchaikovsky



Dying robotic swan brought to you by Mälardalen University.

It's always dangerous to mix robotics with the fine arts, and the latest example is the Dying Swan, created at Sweden's Mälardalen University (**www.mdh.se**) and choreographed by professional dancer Åsa Unander-Scharin. According to a news release, "The Dying Swan is sometimes moving smoothly and gently, sometimes in a dramatic and fiery manner, as Tchaikovsky's majestic music from the ballet Swan Lake is playing ... The swan robot's just over four-minute-long dance has made a big impression. Tearful eyes and words like `touching,' `fascinating,' and `beautiful' are some of the reactions."

Reactions on the Internet, on the other hand, include "scary," "jerky," and "not yet ready for Chuck E. Cheese." Decide for yourself by logging onto **www.mdh.se/news/1.34862**.

Wishy-Washy Robot

Panasonic's hair-washing robot: insert your head and it does the rest.

It's not designed to replace the shampoo monkey at your local beauty shop, and it can't cut hair or do your nails. But Panasonic (**www.panasonic.net**) has introduced a hair-washing robot that may prove useful in hospitals and health care facilities for relieving overburdened medical personnel of a mundane and noncritical task. The machine employs Panasonic's robot hand technology, applying 16 fingers to wash hair and rinse away the little bubbles "with the dexterity of human fingers." Two arms scan the patient's head in three dimensions, and even remember the head shape, allowing the poo-bot to apply proper pressure throughout the process. Each arm has three motors that independently control the swing, press, and massage motions in conjunction with power detection sensors, so it probably does a pretty good job. However, a word to the wise: Make sure it's properly grounded before you stick your head inside. SV

GEER HEAD

by David Geer

Contact the author at geercom@windstream.net

Robots Clean Oil Spills, Climb and Crawl Anywhere

Seaswarm robots and snake robots have been created to service many types of terrains and environments. Between them, they greatly improve the quality of work that robots can perform, cleaning and maintaining a variety of environments.

Seaswarm — One of Practical Robotics Shining Stars

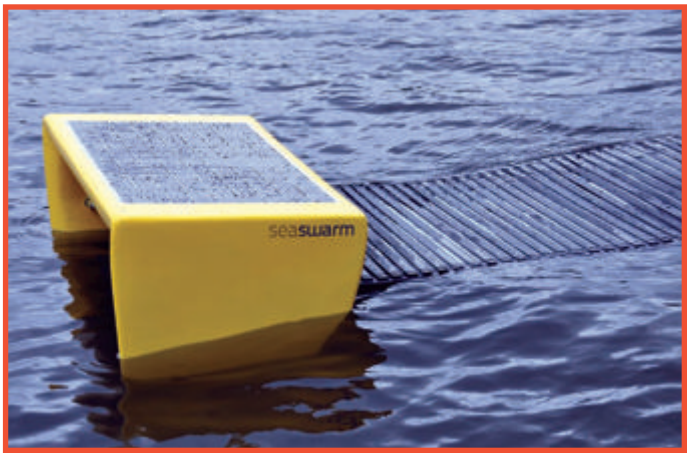
The seaswarm robots are prototyped, autonomous, water-treading swarm robots designed to continually maneuver to oil spills and clean them up by working in tandem. At $25,000 per robot, 5,000 of them — the number necessary to clean a spill the size of the Horizon in one month (according to its creators at Georgia Tech) — would cost $125,000,000. British Petroleum estimated the actual clean-up costs for Horizon were more than $8 billion.

The prototype is 16 feet long by seven feet wide with a yellow head and a black conveyor belt track. Its size enables it to maneuver coastlines and small waterways where larger oil skimming solutions are not small enough to go. The robot is buoyant and the conveyor/track flexes to adjust to the water and the waves. The scientists tested the robot in the Charles River in August '10 where it stayed afloat and powered itself via solar cells located on top of its head. The robot made its public debut at the Venice Biennale's Italian Pavilion in late August. The Biennale is an international festival with participants representing the arts and architecture. The festival organizers devoted it to nanotechnology and how it will change our lives by 2050.

The robot collects oil in a paper-like nanofabric on the very track that drives it through the water. MIT associate professor Francesco Stellacci of the Department of Materials Science and Engineering discovered the material. The smallest components of the material are nanowires which are strewn together to create a porous membrane that absorbs oil at 20 times its own weight while repelling water through the use of a water-repellant coating. The material's properties enable it to absorb and separate oil from water.

Because the material can be heated more than enough to burn off the oil trapped within it without any risk of harm to the fabric itself, scientists can apply it to collecting oil, separating it from water, and burning it off. The fabric makes this novel oil spill clean-up robot possible.

The seaswarm robot prototype in the Charles River; note its yellow head with drive motor and heated oil dissipation inside.



The researchers manufacture the nanofabric in a manner similar to the process of producing wood-based paper. Because the material can separate many other types of elements and compounds from water, users can apply it to many types of water clean-up scenarios.

Light It Up

The environment's powerful new robot ally rids oil by one of three methods, actually still under development. The most likely method is simply burning it off as the fabric passes through a heat chamber in the head of the robot. Researchers construct the fabric with potassium manganese oxide-based nanowires that easily survive temperatures above oil's boiling point. The oil evaporates quickly as a result, according to Adam Pruden, a seaswarm representative. With the burning method, the material could stay in the water in continual use for weeks at a time. Alternate oil processing methods under consideration include digestion by microorganisms or squeezing the oil out of the nanofabric and bagging it for later retrieval.

Researchers power the seaswarm robots using solar cells on top of the head of the robot. Solar panels two meters by two meters covering each robotic head could supply the 300W of power necessary to run the bot. With this amount of power, the robot could do all its processing and move through the water at a rate of 10 kilometers per hour.

The robot could even store surplus energy in its onboard batteries so the robot could continue to move and process oil throughout the dark hours of the night. As solar cells become more powerful in coming years, researchers could make the seaswarm power scenario more efficient.

Sensing and Communications

Seaswarm robots work together, and for that they need a communications modality. The robots are completely autonomous and use a variety of sensor technologies to attend to all their duties. The robots count on precise oil spill location images sent from satellites via Wi-Fi, as well as their own onboard sensors to locate the oil. They count on onboard GPS to tell themselves and each other where each robot is located in relation to the others. The robots sense their water bound surroundings as a group rather than individually. In this way, the sensing capabilities of all the robots combine to tell each robot where to go. The robots also send images they take using onboard cameras as streaming video back to the scientists via the web.

The robots intelligence — comprised of very mature algorithms — works to coordinate the search patterns the robots use to find the oil. The robots can tell the oil apart from other matter by sharing sensor data. To do this, the next seaswarm prototype may use any of the following sensing options: Garmin USB (GPS), Pontech (embedded controllers), Seiko PS 050 (servos), fit-PC2 (computer), Arduino (boards), or MOOS (software).

Clean-Up

In an actual clean-up scenario, the robots would start at the outer edge of a spill location and move inward until all the oil is cleaned from that site. They would then move on to the next spill site and follow the same protocol.

Robot Snakes

Robot snakes are particularly significant because of their form factor and means of locomotion which enable them to navigate tight spots such as pipes and unusual surfaces that other robots would stumble over. Places such as sewers with muck and other substances, as well as trees or inside vertical pipes are easily scaled by robot snakes.

CMU has been experimenting with five or more snake robots that have different gaits, purposes, and levels of development. The scientists achieve the gaits by applying input to the different joint angles to make them move in a certain way, to provide locomotion of a certain manner on a given course. (Think biological gaits.)

Default gaits don't suit every situation. So, gaits are fine-tuned or scripted on a case by case basis to address the difficulty of the area the snakes must scale and the limitations of the snake such as lack of motor strength for specific tasks. Tasks requiring special gates include crossing gaps, reaching or climbing into a hole in a wall, climbing stairs, crossing railroad tracks, and scanning an area using

Snake robot climbs a tree to get a better look (extension cord shown).

the camera in the head of the robot.

Gait Descriptions

Researchers imbued the snake robots with many different gaits that enable them to do some pretty amazing things. The linear gait uses sine waves which the snake robot communicates throughout its mechanisms making it move straight forward or backwards in an up and down kind of motion. The snakes can use this gait in combination with other gaits such as cornering to achieve progress through difficult spaces. The gaits help the robot move through tight areas such as pipes.

A side-winding gait reminds us of rattlesnakes that move sideways. A side-winding robot uses a vertical and a horizontal sine wave to make the snakes move to one side. By side-winding one half of the snake one way and the other half the other way, the snake will roll in place as it moves. This helps it to scale difficult terrain.

With the corkscrewing gait, the robot moves in a spiral that starts at the front of the snake and moves backwards, pushing the snake robot forward. This gait helps the snake move forward and even backwards, especially in spite of obstacles. This gait also helps when the robot is moving through a hole in the wall.

Seaswarm images courtesy of: A project by the MIT Senseable City Lab **senseable.mit.edu/seaswarm, <http://senseable.mit.edu/seaswarm/ss_press.html>**

The remaining gaits include rolling for sideways movement, swimming on top of the water, climbing inside channels and pipes, up poles, around corners, and inside pipes.

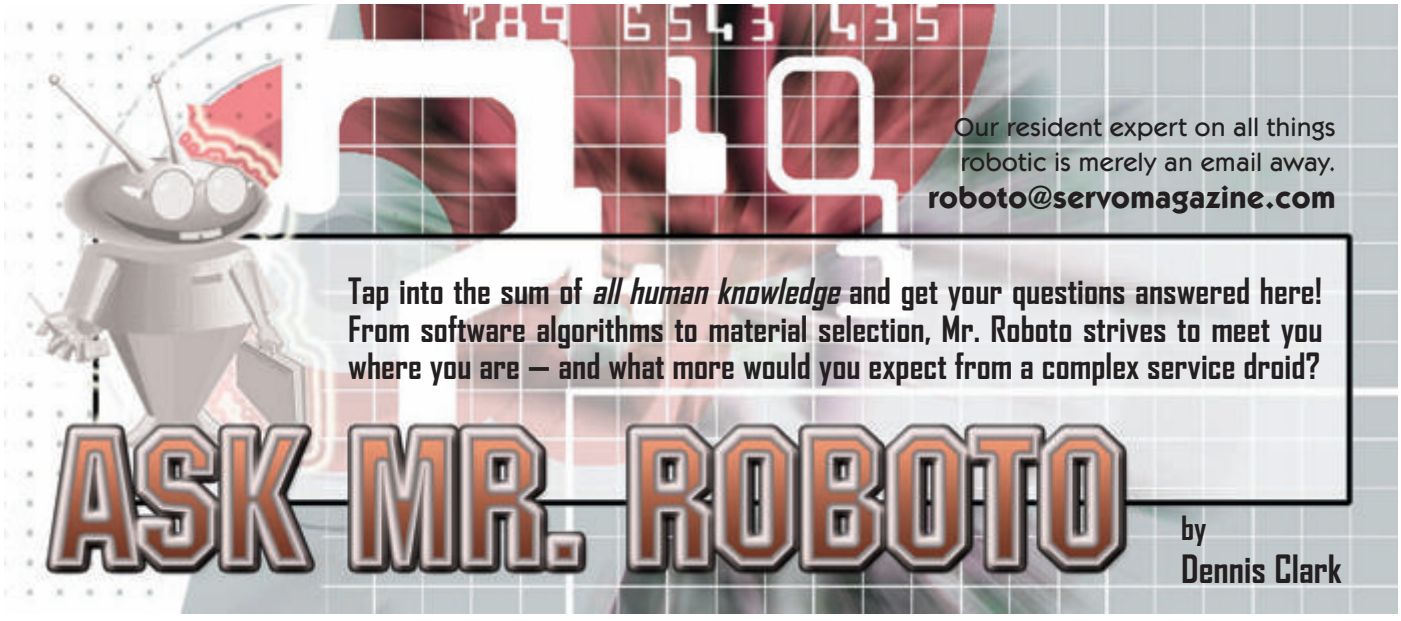
Conclusion

The applications of these robots are immediately practical. While the seaswarm robots have potential for cleaning many substances out of water relatively cheaply, the snake robots can explore, examine, and perform many other functions in many types of environments. Both types of bots can achieve something that eventually touches all of us: a better quality of life. SV

Resources

Seaswarm site
http://senseable.mit.edu/seaswarm

Modular snake robot site
www.cs.cmu.edu/~biorobotics/projects/modsnake



Our resident expert on all things robotic is merely an email away.
roboto@servomagazine.com

Tap into the sum of *all human knowledge* and get your questions answered here! From software algorithms to material selection, Mr. Roboto strives to meet you where you are — and what more would you expect from a complex service droid?

ASK MR. ROBOTO

by
Dennis Clark

By now, First LEGO League's qualifiers will be over and our team will or will not be advancing. Since my crystal ball is in for repairs, I can't say how that will turn out, but my fingers are crossed and the team is working hard. Getting ready for a robot competition is a real ordeal for me and time has flown. I end every competition with grand plans for the next year ... a solemn vow to get started right away. As usual I start working on the robot a month before the next event. Sigh, I never learn. The world is full of distractions, but there is always NEXT year!

I'm going to pick up where I left off last month discussing Finite State Machines (FSMs) and creating the illusion of multiple programs/functions/behaviors running at the same time. I'm going to combine this discussion with my love of hacking toys into robots. Since my mechanical construction skills lag somewhat from my programming and electronic skills, I like to use platforms that someone else has already worked on to get everything to hang together mechanically. The example platform I'm going to use is an old spider that Wowee made several years ago called the "Cyber Spider."

If you can find one on eBay or at your local thrift store, get one. It is supremely hackable. Not long ago, I revisited this creation and added some functionality that attempts to keep it from walking off of a table top. It worked, sort of. Sort of because it requires a rough table top so that the IR beam doesn't reflect completely away from the robot. (Sigh, there are dreams and then there is reality.) Regardless, the code from this project shows very nicely how to get multiple behaviors to work together and simulate an operating intelligence in our mechanical marvels. Onward!

Figure 1. High level FSM.

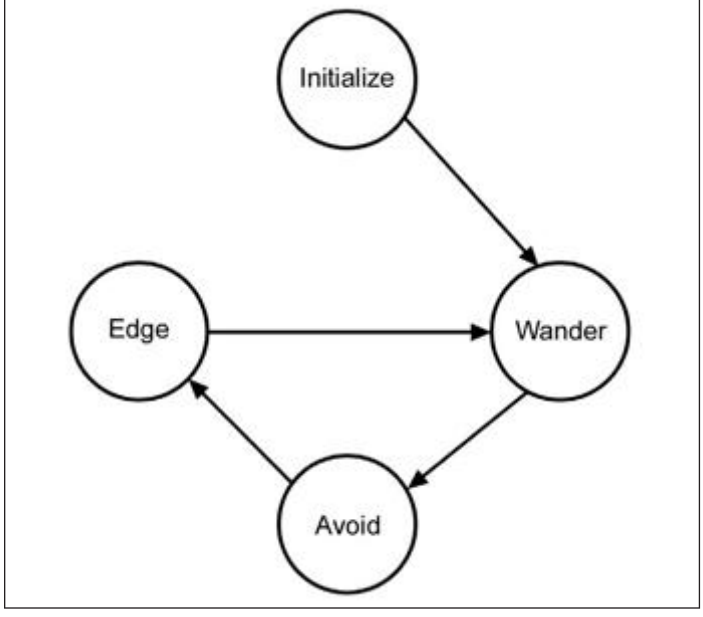
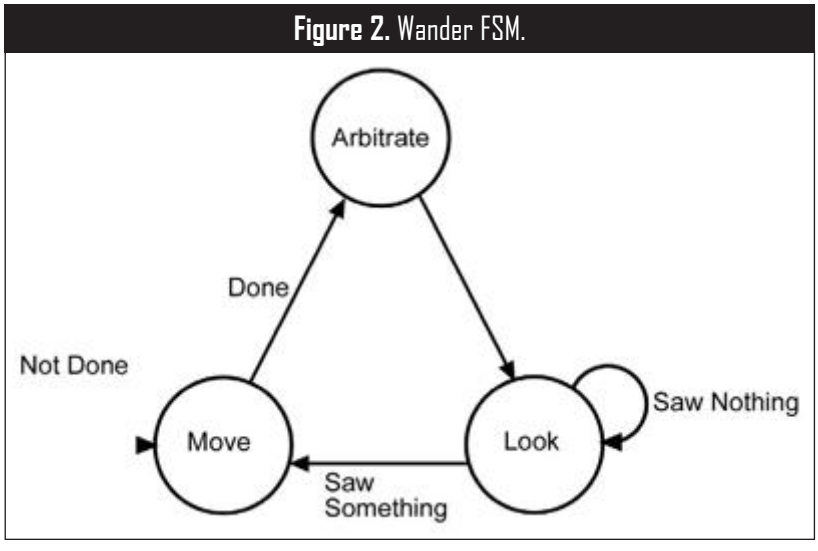


Figure 2. Wander FSM.



Our High Level FSM (Finite State Machine)

Our spider robot has a couple of sensors and

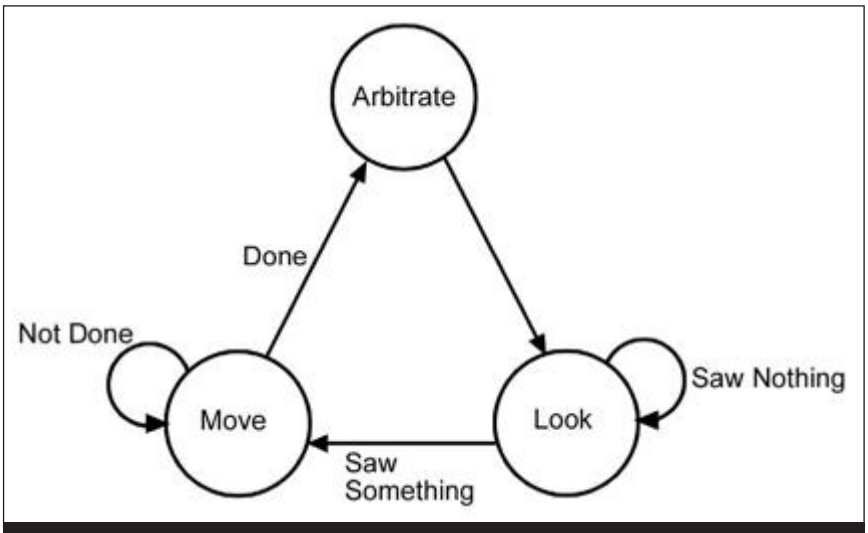


Figure 3. Avoid FSM.

two motors. Let's say we want to have three behaviors operating at different priority levels:

- Wander (so we look curious)
- Avoid (so we don't run into things)
- Edge (so we don't walk off the edge of a table)

To this, we need to add a routine to feed the motors and time what they are doing so that we can go forward, reverse, and turn.

Let's look at our top level FSM (**Figure 1**) to see how things are organized. I'll admit this doesn't look too impressive, but let's expand our scope and look at the FSMs for each of these "behaviors." **Figure 2**, **Figure 3**, and **Figure 4** show the state machine breakdown of each of the behaviors.

The Wander FSM is really simple; just pick a direction and go that way – typically straight ahead until it "sees" something. Not complex, but in my demonstrations kids will find plenty of things to put in its way!

There isn't much to the Avoid FSM. It looks at an IR proximity detector and "does something" if it sees anything. The movements are simple: "turn left," "turn right," and "back away to the right" for some set time.

The Edge Detection FSM is more elaborate. If it sees the table edge, it will back up and turn about 90 degrees or so away before being happy.

How To Code Such Things

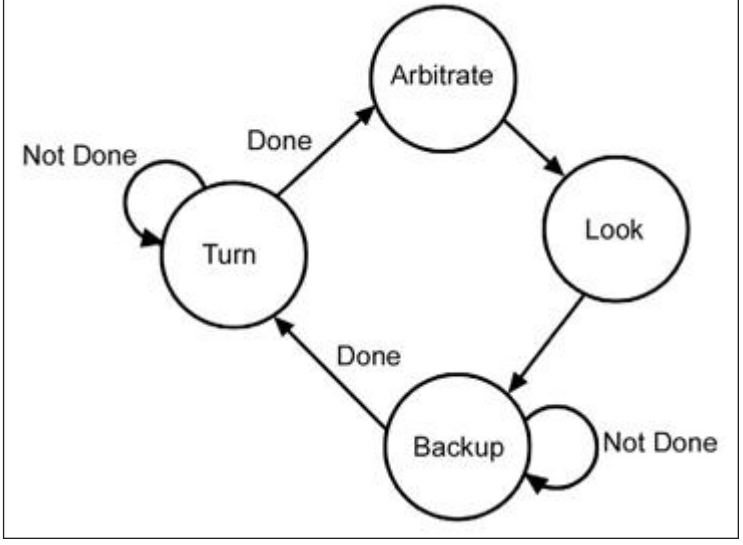
The code is remarkably simple to handle multiple FSMs. To work with several behaviors, the key is to assign each of them a priority. If they don't each have distinct priorities, it will not be possible to determine which one will be active and have control of the motors. I supplied my behaviors in their respective priorities: Wander is the lowest and Edge Detect is the highest.

To make this robot, I used the venerable Microchip PIC16F84 microcontroller. I had a few lying around when I went scrounging. These days, there are a LOT of micros with more capability, but this project was simple. I used the PIC and a 754419 one Amp H-bridge to drive the motors. My sensors were an IRPD board that I developed a long time ago that gives indications of left/right/center as outputs, and finally a Sharp GP2Y0D340K fixed distance IR detector to look down and a bit forward at the table. The source code and comments in the **Init()** function show where things are connected. This is a very simple robot.

I used the CCS PCM compiler for mid-range PICs to write this program. It is simple and does a nice job of abstracting the hardware interface for those new to programming by supplying good hardware setup functions that take away some of the guesswork folks run up against when writing embedded programs. **Listing 1** shows how I assigned priorities and how I implemented my motor control system as an Interrupt Service Routine (ISR).

The interrupt is designed to fire every 32 µs and to count down 16 before starting again. This simulates a 2 kHz PWM to the motors that has 16 settings. This is plenty for the speeds *Slow, Medium,* and *Fast*. The behaviors set the **sleft** and **sright** values used by the motor PWM interrupt service routine. Those routines that set the motor direction and duration use a primitive timing that simply decrements the **dur** variable. The PIC16F84 is running at 4 MHz since the PIC has a four-stage instruction pipeline this means that the *effective* clock speed is only 1 MHz —

Figure 4. Edge detection FSM.



Listing 1: Bugbot setup and motor ISR.

```
//bugf84
//This is the driver program for the Cyber
//Spider.  It uses a 754410 driver for the two
//motors, a TTT IRPD board for object detection
//and will generate a PWM signal for the motor
//drivers. It's designed to use the default IRQ
//handler (very slow) and a 4MHZ 16C84 (also
//very slow) so the PWM resolution is only
//4 bits (16 values in each direction).  Low,
// but more than adequate for this project!
//Version 1.0 © Dennis Clark and TTT Enterprises
//2/2002
//Version 2.0 © Dennis Clark and TTT Enterprises
//1/2009 for 16F628



//#include <16F84.H>
#include <16F628.H>

#fuses HS,NOWDT,NOPROTECT,NOPUT

#use delay(clock=4000000)
#use FAST_IO(A)       //don't keep setting TRIS
#use FAST_IO(B)
#byte PORTB = 6       //writing bits to 754410
                      //direction
#byte PORTA = 5       //IRPD on low bits,
                      //PWM on high bits
#define RTL 0x65      //rotate left
#define RTR 0x6A      //rotate right
#define REV 0x69      //go reverse
#define FWD 0x66      //go forward
#define D_ON 0x2F     //downlook on
#define I_ON 0x4F     //IRPD on
#define B_ON 0x6F     //Both on
#define WANDER 1      //Behavior priorities
#define AVOID  2
#define EDGE   4

const byte SLOW = 6; //slow motor speed
const byte MED = 10;
const byte FAST = 15;

byte IRPDMask;        //masking off bits
                      //controlling IRPDs
byte currDir;         //current direction bits

byte left;            // current left PWM count
byte right;           // current right PWM count
byte PWMperiod;       // to get full 1KHz period

byte sleft;           //left PWM setting
byte sright;          //right PWM setting
const byte SPERIOD = 15;    //16 passes = 2KHz
                            //PWM


//Globals
long dur;             //how long to go that way
byte priority;        //currently active priority

#int_rtcc             // This function is called
                      //every time
void clock_isr(void)
{                     // the RTCC (timer0)
                      //overflows (255->0).
     if (left == 0)
     {
          output_bit(PIN_A2,0);
     }
     left-;
     if (right == 0)
     {
          output_bit(PIN_A3,0);
     }
     right-;
     if (PWMperiod == 0)
     {
          PWMperiod = SPERIOD;
          left = sleft;
          right = sright;
          output_bit(PIN_A2,1);
          output_bit(PIN_A3,1);
     }
     PWMperiod-;      //C int maint makes this >
                      //32us!
     set_rtcc(240);   //255-239 =16, 16*2us =
                      //32us
}
```

which is pretty slow. This allows us to use a simple variable countdown for timing since the numbers don't get very large (the largest one that I use is a countdown of 9,000.) Now, for what you've been waiting for — the code for the behaviors! **Listing 2** has all the rest of the source code for the entire robot. It isn't very complex, and it works great!

Did you notice that the main() function was very "thin?" All of the work is done in the behavior functions and the motor control ISR. To add more behaviors, you simply assign the appropriate priority in the defines at the top of the program and add the function to the **while(1)** loop in main(). The behaviors will tend to interact, but that adds to the fun of seeing what is called *emergent* behavior in your robot (which simply means that it does something that you didn't expect it to do).

Where To Go From Here

I've been wanting to add a photo-phobic or photo-tropic behavior to the Bugbot that would make it either avoid or seek light. This would give some direction to its wandering and make it more fun to play with as an autonomous robot. Can you guess how that would be done? I'll bet you can now!

What We Learned

In this and last month's column, we learned how to create behaviors in robots that interact constructively with each other, and how to appear to run many functions at the same time that won't require a complete recode of your program every time you want to add a new function. This FSM technique can be used in any programming language and is at the core of all embedded programming that doesn't use the bigger fancy real time kernels. You'll find all kinds of uses for this stuff. Trust me. Once you get used to thinking state machines, your life as a robot programmer will get a LOT easier.

So, go out there and build more robots, and add FSMs

Listing 2: The rest of the Bugbot code.

```
//behaviors

void do_wander(void)
{
     //Does just the wander routine

      if (priority == WANDER) {
         dur—;
         if (dur == 0)
             priority = 0;
     }
     if (priority < WANDER) {
         priority = WANDER;
         currDir = FWD;
         PORTB = currDir;
         sleft = MED;
         sright = MED;
         dur = 100;
     }
}

void do_avoid(void)
{                    //Does the object avoidance

     byte irpd;      //IRPD inputs

     if (priority == AVOID)
     {
       dur—;
       if (dur == 0)
       {
             priority = 0;
       }
     }
     if (priority < AVOID)
     {
       currDir = FWD;
       PORTB = currDir;     //Note IRPD bits
         irpd = PORTA;
         irpd &= 0x03;
         if (irpd != 0)
         {
              priority = AVOID;
             switch (irpd)
             {
              case 3:       // directly ahead
                     currDir = REV;
                     PORTB = currDir;
                     sleft = FAST;
                 sright = SLOW;
                     dur = 3000;
                     break;
              case 2:       // something to the
                            // right
                     currDir = RTL;
                     PORTB = currDir;
                     sleft = FAST;
                     sright = FAST;
                     dur =300;
                     break;
                     case 1:       // something
                                   // to the
                                   // left
                     currDir = RTR;
                     PORTB = currDir;
                     sleft = FAST;
                     sright = FAST;
                     dur = 300;
                     break;
             }
         }
     }
}

void do_edge(void)
{
     // Watches for the edge of the table
     byte edet;       //edge detector

     if (priority == EDGE)
     {
       dur—;
       if (dur < 6000)
       {
              currDir = RTR;
              PORTB = currDir;
       }
       if (dur == 0)
       {
             priority = 0;
       }
     }
     if (priority < EDGE)
     {
         edet = PORTB;
         edet &= 0x10;
         if (edet)   //1 == edge of the world
         {
              priority = EDGE;
              currDir = REV;
              PORTB = currDir;
              sleft = FAST;
             sright = FAST;
              dur = 9000;
       }
     }
}

void Init(void)
{
     // This starts it all off, set up the
     // ports, IRQ and defaults
     /*
     PORTA 0 = Left IRPD
         1 = Right IRPD
         2 = Left PWM
         3 = Right PWM
         4 = N/C
         5 = N/C

     PORTB 0 = 1A Right Motor direction pin
         1 = 2A Right Motor direction pin
         2 = 3A Left Motor direction pin
         3 = 4A Left Motor direction pin
         4 = Down Sharp GP2Y0D340K
         5 = Down IRPD enable N/C
         6 = IRPD enable N/C
         7 = N/C
     */

     set_tris_A(0x03);       //0,1 inputs, 2-4
                             //outputs

     setup_comparator(NC_NC_NC_NC); //turn
                                    // comparator
                                    // off

     //Set up PWM interrupt stuff
     setup_timer_0( RTCC_INTERNAL | RTCC_DIV_2);
     enable_interrupts(INT_TIMER0);
     enable_interrupts(GLOBAL);
     set_timer0(240);

     set_tris_B(0x90);       //0-3 outputs, 4&7
                             //inputs, 5&6
                             //outputs
}

void main(void)
{
     Init();
```

Listing 2: continued.

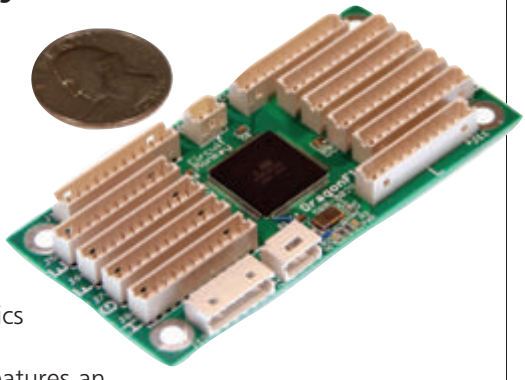
```
    sleft=0;          //left legs speed 0-15
    sright = 0;       //right legs speed
    PWMperiod = SPERIOD;    //overall period
                            //(about 0.5ms)

    currDir = FWD;
    PORTB = currDir;
    sleft = 0;
    sright = 0;

    priority = 0;    // no initial behavior

    delay_ms(1000);  // Lets wait for a few
       // seconds first

    while(1)
    {
        do_wander();
        do_avoid();
        do_edge();
    }

}
```

to your programming toolbox! The source for the Bugbot program can be found at the *SERVO Magazine* website (**www.servomagazine.com**) under *Mr. Roboto* as *Bugbot.zip*. As always, if you have a question for Mr. Roboto, drop me a line at roboto@servomagazine.com and I'll be happy to work on it. Until next time, keep on building those robots! SV

NEW PRODUCTS

DEVELOPMENT BOARDS

DragonFly Controller Board



DragonFly™ is Circuit Monkey's ArduinoMega-capable Atmel ATmega1280 based microcontroller board for robotics and embedded developers. It features an ATmega1280 16 MHz processor with 86 digital I/O lines, 128K byte self-programming Flash program memory, 8K byte SRAM, 4K byte EEPROM, and 16 channel 10-bit A/D converter. It is capable of driving up to 16 RC servos or other PWM devices and is 3.3–5.5V powered. On-board LEDs include one power indicator and one programmable indicator. It is capable of communicating via RS-232, I²C, and SPI. There are four on-chip USARTs. Connectors are equipment-grade reliable Molex MicroBlade connectors. All I/O lines are pinned out to 11 11-pin connectors (each supplying up to eight I/Os plus one power and two grounds). There is a crystal based 16 MHz clock. The board measures 80 mm x 40 mm (3.2 x 1.58 inches). It is software programmable with Atmel AVR Studio or free open-source (avr-gcc, avrdude) tools (not included), as well as the Arduino software IDE. The basic board costs $59.90. An Arduino bundle with cables and USB-serial adapter is available for $89.90. A low weight (11 grams), super thin (2.7 mm), 'naked' version suitable for UAV design is available for $39.90. Other accessories are available individually.

For further information, please contact:

Circuit Monkey Website: **www.circuitmonkey.com**

PLATFORMS

EVALBOT Evaluation Platform

Delivering a fun platform for learning and evaluating real-time software and Stellaris microcontrollers, Texas Instruments Incorporated has announced the availability of its new Robotic Evaluation Board (EVALBOT) for use with Micrium's uC/OS-III. The evaluation kit is a mini robot that allows developers to experience the Stellaris ARM® Cortex™-M3-based LM3S9B92 MCU in real-world applications that leverage the processor's integrated 10/100 Ethernet MAC/PHY, USB On-The-Go, CAN, and motion control capabilities. Based on a complete analog and embedded processing signal chain from TI, the kit includes all of the hardware and software required for quick assembly so that developers can begin evaluation in 10 minutes or less.

Stellaris EVALBOT and book bundle features and benefits (EKB-UCOS3-BNDL) include:

- 80 MHz Stellaris LM3S9B92 MCU with 256K Flash, 96K SRAM, StellarisWare® software in ROM, as well as integrated Ethernet, USB On-the-Go (OTG)/Host/Device and CAN.
- Two DC gearmotors that provide drive and steering, opto-sensors that detect wheel rotation with 45 degree resolution, and sensors for "bump" detection.
- TI motor drivers, voltage regulators, audio codec, interface, and logic devices for easy evaluation of the complete signal chain.
- Bright 96 x 6 blue OLED display and on-board speaker.
- Integrated In-Circuit Debug Interface (ICDI) requires only a single (included) USB cable for software debugging, Flash programming, and serial port connectivity.
- Two 20-pin headers enable future wireless communications using standardized TI low-power embedded radio modules.
- Stellaris-specific version of *uC/OS-III: The Real-Time Kernel* written by Jean J. Labrosse includes example display, audio and motor control projects for EVALBOT, and putting concepts into practice to expedite a user's proficiency.
- EVALBOT robot operates on three AA batteries included in kit.

The Stellaris Robotic Evaluation Board plus Micrium's uC/OS-III book package is priced at $199 US. The Stellaris Robotic Evaluation Board is priced at $149 US and can be purchased separately from the book.

For further information, please contact:

Texas Instruments Website: **www.ti.com**

Is your product innovative, less expensive, more functional, or just plain cool? If you have a new product that you would like us to run in our *New Products* section, please email a short description (300-500 words) and a photo of your product to:
newproducts@servomagazine.com

MOTOR CONTROLLERS

Programmable 2 x 150A DC Motor Controller

Roboteq, Inc., introduces an intelligent controller capable of directly driving two DC motors up to 150 amps each at up to 50V. The HDC2450 is targeted at designers of mobile robots, Automatic Guided Vehicles (AGVs), or any other high power motor control application.

The controller accepts commands from either analog joysticks, standard R/C radio, USB, or RS-232 interfaces. Using the USB or serial port, the HDC2450 can be used to design fully or semi-autonomous robots by connecting it to single board computers, wireless modems, or WiFi adapters.

The HDC2450 incorporates a Basic language interpreter capable of executing over 50,000 Basic instructions per second. This feature can be used to write powerful scripts for adding custom functions, or for developing automated systems without the need for an external PLC or microcomputer.

The controller's two channels can be operated independently or combined to set the direction and rotation of a vehicle by coordinating the motors on each side (tank-like steering). The motors may be operated in open or closed loop speed or position modes with a 1 kHz update rate. The HDC2450 includes inputs for two quadrature encoders up to 250 kHz for precise speed and traveled distance measurement.

The HDC2450 features intelligent current sensing that will automatically limit the power output to 150A in all load conditions. The controller also includes protection against overheat, stall, and short circuits.

The controller includes up to 11 analog, 19 digital, and six pulse inputs. Eight 1A digital outputs are provided for activating lights, valves, brakes, or other accessories. The controller's operation can be optimized using nearly 80 configurable parameters such as programmable acceleration or deceleration, amp limits, operating voltage range, use of I/O, and more.

For further information, please contact:

Roboteq, Inc.
Website: **www.roboteq.com**

GEARBOXES

Toughbox Mini

AndyMark is now offering a brand new Toughbox Mini. This gearbox uses the same interior parts as their standard Toughbox and is shipped unassembled. The CIM motor is not included with this gearbox.

Toughbox Mini accepts two (or one) 2.5 inch CIM motors provided in the FIRST Kit of Parts. Hardware is included for attaching two of these motors.

Specifications

- Overall Ratio: 12.75:1 (standard).
- Gear Material: 4140 steel.
- Gear Type: Spur gears, 20 dp, 14.5 degree pressure angle.
- Gearbox Housing Material: Nylon 6/6 with long fiberglass fill.
- Output Shaft: 1/2 inch diameter with 1/8" keyway, 4140 steel.
- Weight: 1.95 pounds.
- Input Gear: CIM gear, 14 tooth, 0.315 inch bore with 2 mm keyway.
- Large Cluster Gear: 50 tooth, 3/8 inch hex bore.
- Small Cluster Gear: 14 tooth, 3/8 inch hex bore.
- Large Output Gear: 50 tooth, 1/2 inch hex bore.

Optional Gear Ratios

(The standard gear ratio is 12.75:1)

- Optional Ratio 1: 5.95:1, using the 40 tooth output gear and the 24 tooth cluster gear.
- Optional Ratio 2: 8.45:1, using the 45 tooth output gear and the 19 tooth cluster gear.

Weight Reduction With Optional Parts

By using optional parts, a weight reduction of 0.56 lbs is available.

- Aluminum large cluster gear can replace the standard steel 50 tooth large cluster gear.
- Aluminum large output gear can replace the standard steel 50 tooth large output gear.

Additional Encoder

An optional encoder package can be purchased and added to this Toughbox. If you already have a US digital encoder to fit this Toughbox, the encoder mount pad will help you mount the encoder to the Toughbox.

For further information, please contact:

AndyMark, Inc.
700 E. Firmin St., #114
Kokomo, IN 46902
Website: **www.andymark.com**

EVENTS Calendar

ROBOTS.NET

Send updates, new listings, corrections, complaints, and suggestions to: steve@ncc.com or FAX 972-404-0269

Know of any robot competitions I've missed? Is your local school or robot group planning a contest? Send an email to steve@ncc.com and tell me about it. Be sure to include the date and location of your contest. If you have a website with contest info, send along the URL as well, so we can tell everyone else about it.

For last-minute updates and changes, you can always find the most recent version of the Robot Competition FAQ at Robots.net: **http://robots.net/rcfaq.html**

— R. Steven Rainwater

DECEMBER

2-5 **ROBOEXOTICA**
Vienna, Austria
Robots compete at serving cocktails, bartending conversation, and other achievements in electronic cocktail culture.
www.roboexotica.org

3-4 **Robotex**
Tallinn University of Technology, Tallin, Estonia
Autonomous robots compete on a field where they must move multiple balls to goals.
www.robotex.ee

4 **FIRST LEGO League of South Africa Championship**
South Africa

Regional South African FLL teams compete for the National Championship.
www.fllsa.org.za

14-17 **IROC International Robot Olympiad**
Queensland, Australia
Robots compete in a wide range of events including robot dancing, push-out, robot survival, robot prison break (wheeled and legged versions), non-programmed line tracer, non-programmed race, robot in a movie, mission challenge, and the cart rolling-ball maze solving challenge.
www.iroc.org

JANUARY

11 **First LEGO League of Central Europe**
Heinz Nixdorf Museum Forum
Paderborn, Germany
Regional European FLL teams compete for the National Championship.
www.hands-on-technology.de/en/firstlegoleague

25 **Powered by Sun**
Ostrava, Czech Republic
Solar powered autonomous robot race.
http://napajenisluncem.vsb.cz

25-27 **Singapore Robotic Games**
Singapore Science Center, Republic of Singapore
Autonomous robots compete in 17 events that include Sumo, legged robot marathon, legged robot obstacle race, micromouse, underwater, and more.
http://guppy.mpe.nus.edu.sg/srg

28-31 **Robotix**
IIT Khargpur, West Bengal, India
Robots of all kinds compete in events including Xants, TribotX, Xtension, 8mileX, ASME, Xplode, and eXplore.
www.robotix.in

MARCH

6-10 **APEC 2011**
Fort Worth Convention Center
Ft. Worth, TX
Autonomous micromouse maze running for bots 25 cm x 25 cm. Cash prizes and trophies.
www.apec-conf.org

bots IN BRIEF

FLAP THIS WAY

Pioneer and iXS research have jointly developed a wacky-looking car navigation robot called "Carnaby."

The robot guides the driver as to whether turn right or left by movement of its wings as soon as it receives the navigational instructions. As you come closer to the turning point, it flaps its wings and its eyes glow.

They are continuing work on the robot to make it helpful for older people, as well as for people with hearing disabilities.

BLOCK HEADS

These cute little blocks are called Cubelets. Each one is a robot with unique programming, capabilities, and behaviors. However, the magic happens when you stick the blocks together and they cooperate to create an entirely new robot.

Each block communicates with its neighbors, so you know that if two blocks are next to each other, they're talking. If you make a simple robot by connecting a Light Sensor block to a Speaker block, they'll start to talk, and when the light in the room gets brighter, the Speaker will get louder. Actually, you'd need a third block to make this work. Every robot needs a Battery block to run.

Next, you could swap the Speaker for a Drive block, and when the light gets brighter, the robot will drive faster. A third category of blocks is the Think Blocks. Maybe you'd want to put an Inverse block in between the Light Sensor and Drive blocks. Then, the robot would drive slower as the light gets brighter. This simple communication between adjacent blocks is what gives the kit that magic.

The basic kit (which you can pre-order now for $300) includes 20 Cubelets:

Action Blocks: 2 Drive, 1 Rotate, 1 Speaker, 1 Flashlight, 1 Bar Graph.
Sense Blocks: 1 Knob, 1 Brightness, 2 Distance, 1 Temperature.
Think/Utility Blocks: 2 Inverse, 1 Minimum, 1 Maximum, 1 Battery, 2 Passive, 2 Blocker.

There are no wires involved, and no programming, so Cubelets are suitable for children as young as 5. Technically, Cubelets are in beta testing, so it'll be exciting to see some of the combinations that people come up with.

LOST ... BUT NOT FOUND

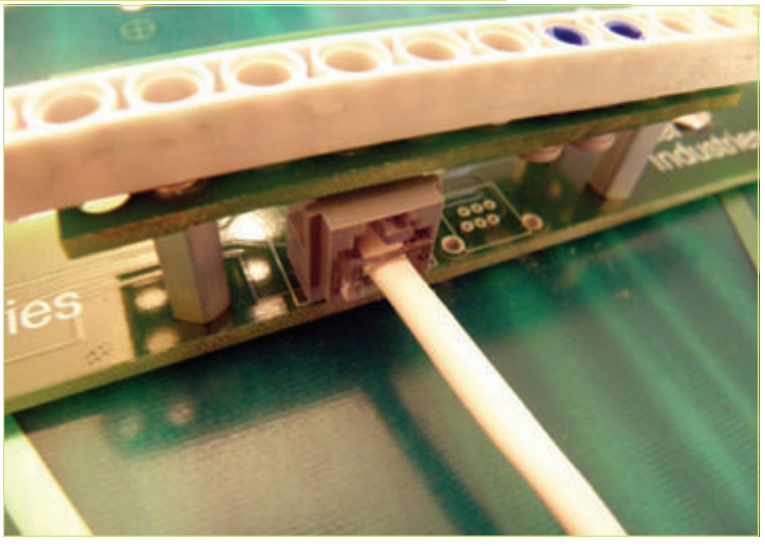
After a hot air balloon went missing over the Adriatic Sea near Italy, the Italian Coast Guard utilized an underwater unit that employed a robot to aid in the search. The balloon — carrying Richard Abruzzo and Dr. Carol Rymer-Davis — was equipped with a satellite phone, a radar transponder, VHF radios, and two mobile phones, at the time of print, the search has only come up with small pieces of debris, unfortunately.

bots IN BRIEF

FUN WITH THE SUN

Now you can run your LEGO Mindstorms NXT using the sun with the Dexter Industries 2W dSolar System that easily attaches to any of your creations. The kit comes with a 9V solar panel, a power cord, and adapter at a price of just $90.

The 9V solar panel provides approximately 250 mA — enough to power the NXT and a single motor in direct sunlight.



ANDROID IN CONTROL

RT Corp and Brilliant Service Co. Ltd. came together to build RIC (Robot Inside Character) — a literal robot controlled by an Android. Encased in a plastic suit, the android can walk, move its hands and head, and plays synthetic sounds and music via wireless LAN. It was recently displayed at the Google Developer Day 2010. Future plans include extending its functions by April 2011 then renting it to other companies for about $6,000 a day to promote other Android-based devices.

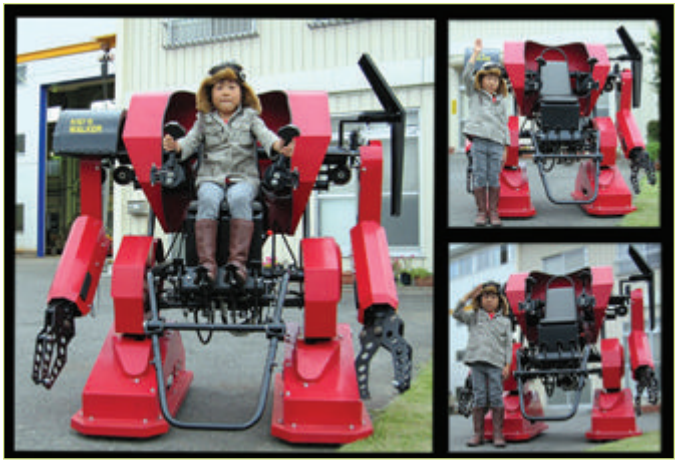
Cool tidbits herein provided by Evan Ackerman at www.botjunkie.com, www.robotsnob.com, www.plasticpals.com, and other places.

STEELING THE SHOW

Filming has now concluded on "Real Steel" — Hugh Jackman's new Sci-Fi movie. The plot involves a former boxer (Jackman) who has to reinvent himself when 2,000 lb, eight ft tall robots take over the sport. He teams with his estranged son to build and train their own oversized Rock 'Em Sock 'Em Robot.

The premise of the movie is actually based on a short story by Richard Matheson that was adapted into an original 'Twilight Zone' episode that starred Lee Marvin. Here's a picture of the original robot.

Oh, and be sure to look for *SERVO Magazine* in the background of some of the scenes.

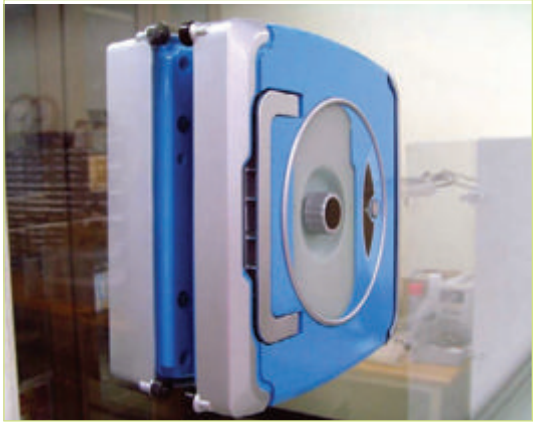


EXO MINI-ME

Check out this mini-exoskeleton from Sakakibara Kikai. At a height of 5.25 ft and a weight of 400 lb, it will strike fear in mere mortal toddlers and more likely, their parents. The $21,000 price tag of the walker doesn't seem to include an adequate seatbelt, however.

ALL WASHED UP

Researchers at the Pohang Institute of Intelligent Robotics (PIRO, South Korea) have unveiled a window cleaning robot called Windoro that will be commercialized next year. The robot — which had a budget of approximately 300M KRW ($260,000 USD) — consists of two modules that hug together on opposite sides of a window using neodymium magnets. The magnetic system is said to be safer and more reliable than other approaches such as vacuum power. The robot uses distance sensors, attitude adjustment, and obstacle detection while doing its window waltz, employing detergent and a series of spinning pads to wash up as it goes.



SOLAR FLAIR

At the recent CEATEC 2010 in Japan, Nissan used the opportunity to show off its latest EPORO in its demo of their 39 ft tall solar tree — a charger that generates 20 kW of power. The robotic car is rigged with a wireless power system and can recharge itself on the tree, as well as charging lanes on the road. Don't get too excited, though. Nissan is targeting 2030 as the year the system becomes commonplace.

Solar trees can be used individually as small-scale charging stations in urban areas or they can be grouped into forests to produce energy on the scale of power plants. According to Nissan's design, a forest of 1,000 solar trees will be able to provide electricity for 7,000 households.

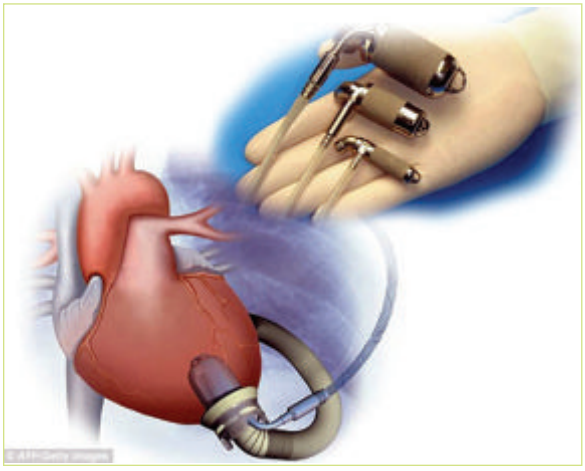
In addition to providing power, solar trees can provide some relief from the heat in summer. The translucent solar panels offer protection from UV light, while fine mist emitted from the edges of the panels works to reduce the temperature in the immediate vicinity.

IT'S A BIRD! IT'S A PLANE! NO, IT'S A ...

This is an honest to goodness, real, working, commercially-available jetpack. Don't get too excited here either. Yep. The Martin Jetpack is now robotic and has no use for humans. Damn, that one piece of the future that would have been totally awesome has just passed us by.

Martin Aircraft Company has actually been working on a practical jetpack for years now. The lowdown is that the jetpack uses two huge ducted fans that you wear kinda like a backpack and it will propel you 8,000 feet up at 60 miles an hour for 30 minutes. It runs on the same gas your car does, doesn't require a pilot's license, and includes a ballistic parachute (that works at low altitudes) — just in case.

From the beginning, Martin had autonomy in mind for their jetpack for one simple reason: Nobody really wanted to be the first guy who had to strap it on and see how fast it would go, or who had to check if the emergency parachute system worked. So, it was a natural step to turn the robotic testing system into a totally robotic flying system. Martin is hoping that their 'Skyhook' will be able to fill a niche between man-portable reconnaissance UAVs and larger, infrastructure-dependent drones like Predators. Skyhook can take off and land vertically while carrying up to 100 kilos of payload which would be ideal for local resupplying of isolated units.



GOTTA HAVE HEART

A 15 year old Italian boy has become the first child patient in the world to be permanently implanted with an artificial heart.

The boy — who has not been named — underwent a 10 hour operation last week and at last report is still in intensive care but has woken up following the surgery and is said to be well and talking.

As he already suffers from a muscle wasting illness called Duchenne syndrome. he was ineligible to be placed on the heart transplant waiting list.

The illness causes rapid muscle degeneration and the teenager had been confined to bed and unable to walk, and was close to death when surgeons decided to install the artificial heart.

Paediatric cardiac surgeon Dr Antonio Amodeo carried out the operation with an eight-member strong team at the Bambino Gesu Children's Hospital in Rome.

HIT ME

Isaac Asimov would probably be horrified at the experiments underway in a robotics lab in Slovenia. There, a powerful robot has been hitting people over and over again in a bid to induce anything from mild to unbearable pain — in apparent defiance of the first law of robotics which states that "a robot may not injure a human being."

However, the robo-battering is all for a good cause, insists Borut Povše, who has ethical approval for the work from the University of Ljubljana, where he conducted the research. He has persuaded six male colleagues to let a powerful industrial robot repeatedly strike them on the arm to assess human-robot pain thresholds.

It's not because he's defying the first law of robotics. It's to help future robots adhere to the rule. "Even robots designed to Asimov's laws can collide with people. We are trying to make sure that when they do, the collision is not too powerful," Povše explains. "We are taking the first steps to defining the limits of the speed and acceleration of robots, and the ideal size and shape of the tools they use, so they can safely interact with humans."

Povše and his colleagues borrowed a small production-line robot (made by Japanese technology firm Epson) normally used for assembling systems such as coffee vending machines. They programmed the robot arm to move towards a point in mid-air already occupied by a volunteer's outstretched forearm, so the robot would push the human out of the way. Each volunteer was struck 18 times at different impact energies with the robot arm fitted with one of two tools: one blunt and round, and one sharper.

The volunteers were then asked to judge for each tool type whether the collision was painless or engendered mild, moderate, horrible, or unbearable pain. Povše — who at least tried the system before his volunteers — said most judged the pain was in the mild to moderate range.

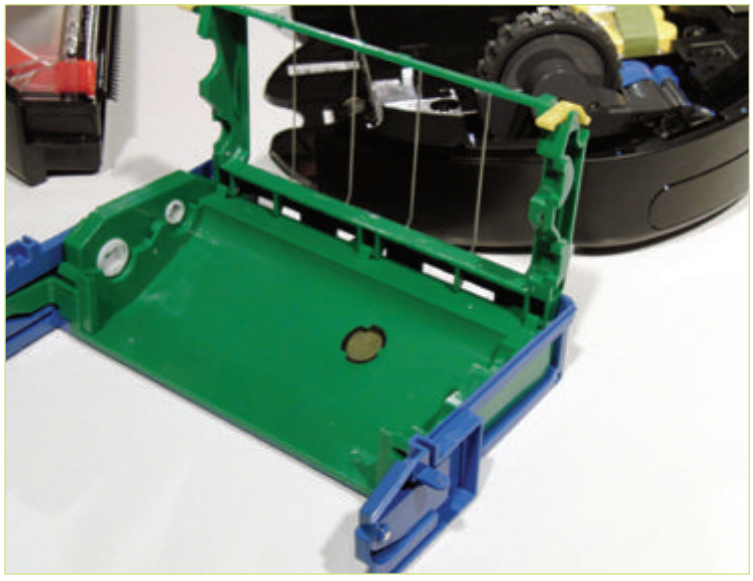
TAKES THE CAKE

Yes, robots are taking our jobs, but it takes a truly evil mind to design a robot to put grandmothers out of work. This robot cake decorator works sort of like a Spirograph; the cake rotates in a circle while an arm moves back and forth depositing frosting. A separate arm drops those little silver balls that you always wonder whether or not you're really supposed to eat them. It was designed by Katherina Mischer and Thomas Traxler.

GET THE DIRT

The newest Roomba version now offers modular parts that can be removed and replaced easily by users. Instead of mapping and memorizing a room — which would take additional technology and raise the price of the Roomba (which starts at $200) — the unit's iAdapt technology uses an internal dirt sensor which literally listens for the sound of dirt being scooped up. With this information, the unit can decide to return to the scene of the dirt to give it additional sweepings.

iRobot introduced its fifth generation Roomba at a recent press event in New York.



WALK TALL

The woman in this picture is Amanda Boxtel, who has had a T11/12 spinal injury for 18 years. She's a paraplegic, but now she's able to walk with the aid of eLEGS — a robotic exoskeleton system from Berkeley Bionics. You probably remember Berkeley Bionics from their cargo-carrying exoskeleton, HULC, which they've since licensed to Lockheed Martin for production for the military. eLEGS is largely based on HULC, except it's designed for (eventual) home use. The system is relatively light at 45 pounds, and you strap into it by yourself while sitting down. After only a few hours of practice, paraplegics are able to use eLEGS to stand up and walk.

COMBAT ZONE

Featured This Month:

Features

Events

PARTS IS PARTS

Fingertech TinyESC v2 Review

● by Thomas Kenney

Just a few weeks back, Fingertech Robotics released the upgraded version of their popular TinyESC single channel speed controller. I was one of the first to have a chance to try out this new piece of hardware in the arena, and my experience so far has been nothing but positive.

The original TinyESC was released to the market in late 2008 and soon became a favorite among insect builders. Two separate versions of the speed controller were made available: one requiring an external 5V from the receiver; and the second with a 5V 100 mAh continuous battery eliminator circuit at no real weight difference and an increased cost of less than $5. It also featured reliable temperature, over-current, and over-voltage protection to help prevent the release of the infamous magic smoke that plagues most electronics in this field. Of course, the most defining attribute of the TinyESC is its namesake compact and lightweight design – take it up less than five grams and is small enough to be crammed inside a half inch cube.

The TinyESC v2 keeps all of

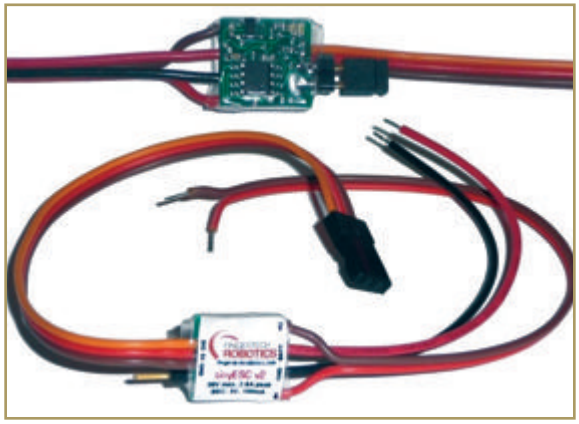


PHOTO 1, A front and rear view of the TinyESC v2 displaying the header pins and jumper used for the speed controller's new calibration function.

www.servomagazine.com/index.php?/magazine/article/december2010_CombatZone

these great features, even expanding on some of them as the 5V battery eliminator is now incorporated into every speed controller at the same price of the first version, sans BEC. What's probably the most prominent new feature on this revision is the addition of a calibration function, setting the trim dead on, and eliminating the need for any adjustment on the transmitter end. The method is simple — it uses the two header pins and a small jumper (both seen on the back of the ESC in **Photo 1**) to move into calibration mode. The calibration function is also accompanied by a new set of status LEDs indicating the calibration process, as well as a directional signal. The only downside with any of these new features is the .5 grams increase in weight and .4" increase in length due to the addition of the calibration header pins. Nonetheless, for the space and weight paranoid (as I myself am with smaller bots), this menial increase can be dealt with easily by trimming the pins after the calibration is set up.

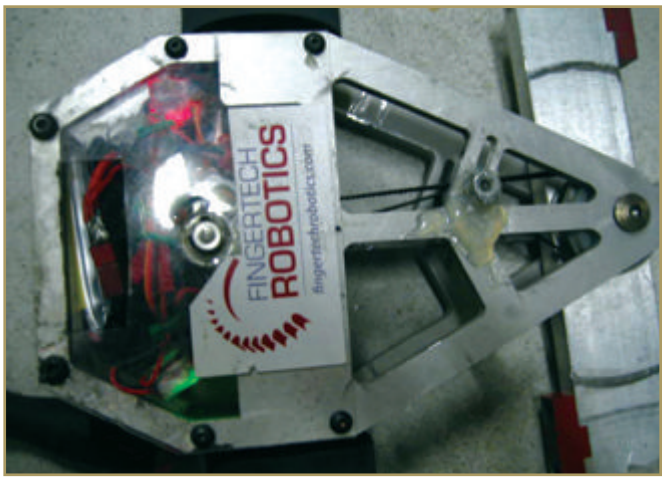


PHOTO 2. Though the robot is a bit bigger than where TinyESCs are usually used, I needed the weight, and the new TinyESCs delivered; they didn't show a hiccup even when running the hot wound B-series motors in a three pound robot!

Overall, I'm very pleased with the second generation of this speed controller. It keeps all the great features of the original and expands on those to create an ESC for small combat bots that's one of the most easy to use, compact, and bulletproof on the market today.

The TinyESC v2 is available from **www.fingertechrobotics.com** and should soon be carried by Fingertech's US distributer at The Robot Marketplace. **SV**

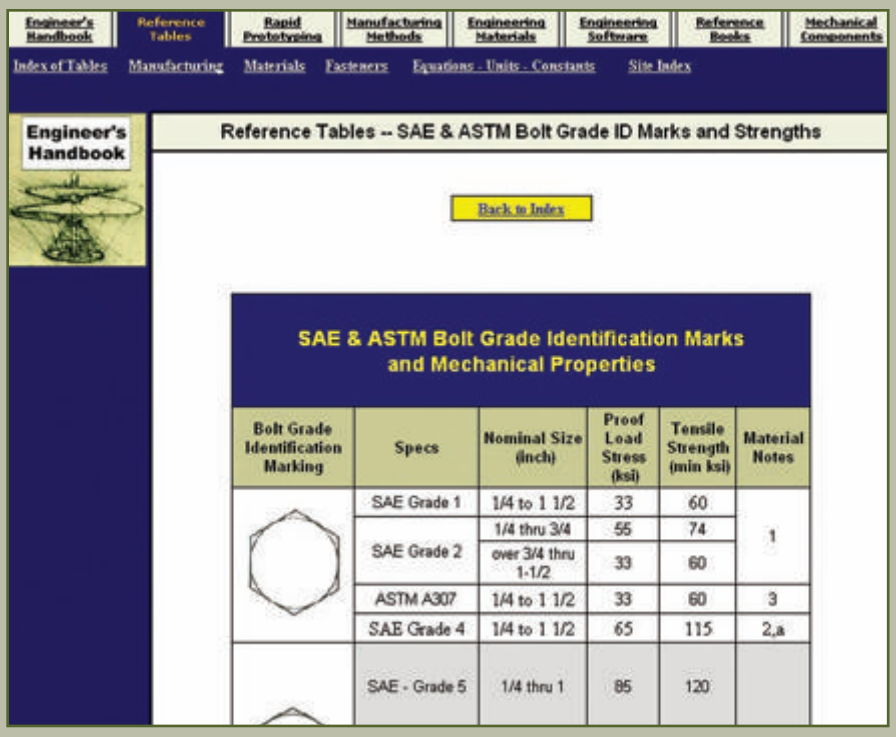
MANUFACTURING: Calculating Bolt Torques

● by Matthew Spurk with Katherine Kelly

Sparks fly as the two robots smash into each other again. Sweat drips down your forehead as you prepare for another charge across the arena. It's been a good battle, but you definitely have had the upper hand landing several solid blows, but your opponent is tough. He takes the blows and keeps coming back for more. The announcer calls out "one minute remaining in the bout." A few seconds later, you notice that your robot is beginning to pull to one side when you try going forward. Oh No! You've lost your right side drive. There's still 40 seconds in the match and now your opponent has a distinct advantage. You struggle to keep the weapon pointed at him, but with 20 seconds left in the match the wheel slips off the end of the motor shaft. Your opponent proceeds to make full box slams across the arena, and you're helpless to stop them. Time expires and it goes to a judge's decision. You enter the arena and disable your bot, then do the walk of shame across the arena to gather that missing wheel. The judges turn in a split decision and it's not in your favor. You manage a valiant shrug, shake your opponent's hand with a smile, and head out of the arena. As you pull your bot back to the pits you hear the calls of "unlucky" and "tough break" from your fellow competitors, but you know, deep down, that they're wrong.

Had you just read that article in *SERVO Magazine* on bolt torques, none of this would have happened. Here is your chance to prevent that from ever happening again!

If you've ever had to work on your car, you've probably come across a manufacturer specified bolt torque. The purpose of that torque is to ensure that the bolt will not work loose over time. As the nut is turned, it creates tension (stretch) in the bolt. This stretching creates a clamping force also known as a bolt preload. What this article explains is how to calculate how much force you need to apply to your wrench in order to achieve that force in the bolt. Calculating the force required to stretch the bolt is a relatively easy calculation provided you know a few key pieces of information about your bolt and its environment.



Engineer's Handbook | Reference Tables | Rapid Prototyping | Manufacturing Methods | Engineering Materials | Engineering Software | Reference Books | Mechanical Components

Index of Tables | Manufacturing | Materials | Fasteners | Equations - Units - Constants | Site Index

Engineer's Handbook

Reference Tables -- SAE & ASTM Bolt Grade ID Marks and Strengths

Back to Index

SAE & ASTM Bolt Grade Identification Marks and Mechanical Properties

Bolt Grade Identification Marking	Specs	Nominal Size (inch)	Proof Load Stress (ksi)	Tensile Strength (min ksi)	Material Notes
	SAE Grade 1	1/4 to 1 1/2	33	60	
	SAE Grade 2	1/4 thru 3/4	55	74	1
		over 3/4 thru 1-1/2	33	60	
	ASTM A307	1/4 to 1 1/2	33	60	3
	SAE Grade 4	1/4 to 1 1/2	65	115	2,a
	SAE - Grade 5	1/4 thru 1	85	120	

Know Your Bolt's Grade

If you want to be the top of your class, then you need to make the grade. Well actually, you need to know the grade. Bolts come in all shapes, sizes, and even colors. You may have seen a bolt that is gold in color at your local hardware store. The sign below that bolt may say "Grade 8." Looking at the color is not a tell-tale sign of determining what grade bolt you have. Looking at the color tells you what color bolt you have. If you want to be certain what grade bolt you have, you need to look closer. There are different markings on the top (head) of the bolt that allow you to determine what grade of bolt you have. The grade 8 bolt mentioned earlier should have six dashes that extend from each point of the head toward the center of the bolt. There are numerous sources on the Internet for bolt grade identification. A personal favorite of mine is **www.engineershandbook.com/Tables/boltgrades.htm**. Along with the grade markings, the referenced table also includes the proof strength of the bolt, explained below.

The Proof is in the Strength

In order to ensure that the bolt achieves the bolt preload described above, the bolt stress must be above 75% of the bolt's proof strength. That statement isn't the easiest to understand, so let's break it down. Let's start with the proof strength. The proof strength of a bolt is the point at which an average bolt will begin to yield, a.k.a., permanently deflect, a.k.a., it stretches and stays there. We need to apply a load to the bolt that is greater than 75% of that load. We want to be close to the point where it stretches permanently, but we also don't want the bolt to yield.

Some people will use 80% or 85%. I prefer to err on the side of caution, and would rather have a nut work loose than have a bolt fail. Knowing the type of bolt that you have, the size, number of threads per inch, the grade, etc., you can calculate your bolt's proof strength or you can do what thousands of engineers do every day and just look it up online. Going back to **www.engineershandbook.com/Tables/boltgrades.htm** will get you that information.

K-factor	Plating	Lubrication
0.11	Zinc Plated	Loctite 262
0.13	No Coating	Anti-seize Nickel
0.15	Zinc Plated	Loctite 242
0.16	No Coating	Anti-seize Copper
0.18	Black Oxide Coated	Dry
0.18	No Coating	Loctite 262
0.2	No Coating	Dry
0.2	Cadmium Coated	Dry
0.2	No Coating	Loctite 242
0.21	Zinc Plated	Loctite 277
0.26	Zinc Plated	Dry
0.26	No Coating	Loctite 277

TABLE 1. Nut-factors for common bolts and lubricants.

Put a Coat On

Remember that gold color we discussed earlier with the grade 8 bolts? That golden color comes from the bolt's coating. That gold color is actually a yellow zinc coating that helps to protect the bolt from corrosion. Most bolts have zinc-plating — either silver or yellow. Bolts can also be galvanized to help protect the bolt from corrosion. There are a multitude of coatings available.

An important consideration in calculating bolt torques is to determine whether or not to include lubrication. In some instances, it may be desirable to ensure the bolt can be easily removed at a later date. Did you put that anti-seize compound on your spark plugs the last time you installed them? You may also want to ensure that the bolt does not come out easily and decide to put a thread locker on the bolt, i.e., Loctite.

The coating and the lubrication are referred to as the Nut-factor, the K-factor, or the Friction Factor. **Table 1** references some common Nut-factor values based on coatings and lubricants. Lubricants or coatings that increase the friction in the system will have a lower value and therefore will require less torque to ensure the bolt remains tight. If you don't know what coating you have, don't sweat it. Many engineers will use 0.2 regardless of the hardware they are using.

$$A_t := \frac{\pi}{4}\cdot(D - .9743\cdot p)^2 = \cdot in^2$$

EQUATION 1. Calculation for tensile stress area for a 1/4-20 bolt.

$$F := .75\cdot\sigma_{proof}\cdot A_t = 2915\cdot lbf$$

EQUATION 2. Calculation for bolt force for a 1/4-20 bolt.

$$\tau_{pl} := F\cdot K\cdot D = 12\cdot ft\cdot lbf$$

EQUATION 3. Calculation of recommended torque for a 1/4-20 bolt (dry, no lubrication).

Don't Get all Mathy on Me

Now we can get into the meat of the calculations. We need to determine the bolt pitch. To get the bolt pitch, take the inverse of the thread count. If you have a 1/4-20 bolt, the 1/4 is the diameter of the bolt and the 20 is the number of threads per inch. To get the pitch you divide 1 by the threads per inch, or, in the case of a 1/4-20 bolt, you have 1/20 or .05.

From there, we will calculate the threaded area of the bolt — which is referred to as the tensile stress area — using **Equation 1**, where D is the bolt diameter and p is the pitch. Next, begin to calculate the Force that is required to maintain the preload using **Equation 2**, where theta sub proof is the proof strength from our reference table and A sub t is the tensile stress area we calculated in **Equation 1**.

Now, calculate the recommended torque to achieve that value using **Equation 3**, where k is the Nut-factor, D is the bolt diameter, and F is the Force from **Equation 2**. This number is the amount of torque you must apply to achieve the preload we discussed in the beginning. If you use American standard units throughout your calculations, this number will be in in-lbs or ft-lbs.

Now to apply that torque, you will need to use a torque wrench or some equivalent method to apply that torque to the nut. You want to avoid applying it to the bolt if possible because the friction between the bolt and the surfaces to be joined may affect your applied force.

What's Next?

There are certainly other factors that can be taken into consideration to optimize a bolt torque, such as: the materials that are being clamped; the number of clamped pieces; and whether a gasket is being used, etc. All of these things will affect how much torque is required to maintain that bolt preload, but people much smarter than me figure that stuff out. Hopefully, you will be able to use this article to ensure that those ever so common wheelectomies become less common. SV

EVENTS

Completed Events September 15th – October 17th

Franklin Institute 2010 was presented by the North East Robotics Club in Philadelphia, PA, on October 9th. Thirty-nine bots were registered.

HORD Fall 2010 was presented by the Ohio Robot Club in Akron, OH, on September 25th. Twenty bots were registered. SV

POTPOURRI

● by Kevin M. Berry

That's right, botheads, I used the word "potpourri" in an article title! Just like a fighting machine, I have a soft vulnerable inside also. Get over it, okay?

We all have that bin in our shops (what my father used to call "the string too short to use drawer"). It's full of orphaned bolts, slightly bent shafts, wads of wire soldered carefully together then cut frantically apart in the pits, and that mysterious "shop dust" composed of rust, dirt, shavings, and (dare I speak the truth?) insect droppings.

This new feature is going to be like that bin. Odd and interesting items gleaned from the RFL forum, my personal research, and reader submittals, hopefully without any bug droppings! No promise that it will appear regularly, but when the bin is full, I'll scrape a layer off the top to share with you. (Ick.)

(Editor's note: In another sign of how sick our society is, Googling "junk drawer" brings up 481,000 images. Googling potpourri brings up over two million. Double ick.)

Useful Online Calculators, Converters, and Tutorials

One of the amazing things about those Internet searches is the amount of pre-digested information available, enabling those of us who grew up using books of tables and slide rules to brainlessly smash data into handy apps, and voila! – instant answers!

Here are a few sites I've encountered lately, not all directly applicable to bot building, but with enough cool factor to publish anyway.

Power transmission and drive engineering calculators (power, torque, force, speed, acceleration, inertia, unit conversions)
http://imtdrc.com/pt-fromula/#

Wind driven generator calculator (power available for given wind speed)
www.awea.org/faq/windpower.html

Team Cosmos kinetic energy calculator
www.teamcosmos.com/ke/ke.shtml

Theory of spinning disks
http://robots.freehostia.com/Spinningdisks/Disks.html

Team Tentacle torque/amp-hour calculator
www.architeuthis-dux.org/torquecalc.asp

Multimeters – Wisdom From Builders

A recent post on the RFL forum led to a great discussion on multimeters:

"I was wondering what would be a good multimeter brand and model to buy. What do you folks think? What features should I look for? Does true RMS matter? Should it read capacitor values? How much amperage should it handle? What is a good price for a good meter?"

As always, the wisdom and experience flooded in. The replies are presented below (somewhat edited for grammar, complete sentences, etc.).

"If you don't have a separate clamp-on ammeter, the Midtronics (MDT100) could do what you need."

"We have used Flukes at my work for years. As a rule, an amp-clamp is so-so if you want accurate flows, so we have used shunts. These are a little more cumbersome to hook up, but nuts-on for reads. I like the Fluke compact multimeter (FLK-115). I believe this model has a freeze mode that will hold the highest/lowest read in display ... a must for those pesky surges."

"The features depend on what you do. If you do component-level troubleshooting, it can be handy to have capacitor checking and transistor/FET checking in one box. I have and prefer separate devices for such tasks. They tend to be a little better quality that way. Any new meter is going to be "true RMS" so I wouldn't get so hung up on that. I'll second the other post on Flukes. If you have the cash, they are a great meter. They just flat out work. (On the other hand, I still use a Simson 260 at work sometimes.)"

(Editor's note: In my job at Kennedy Space Center, many techs tell me they still prefer the Simpson analog meter to any of the digital ones. The visual indication of "open" vs. "continuity" or "voltage" vs. "dead" is so obvious. The needle swings or it doesn't. Plus, any meter that is still ticking after 30+ years of use and abuse has to have something right about it!)

"In the area of budget-minded meters, I've had several of the Harbor Freight brand and most have been just fine. Those are the ones I keep around as "loan outs" in the pits. They get the job done and if they disappear it was only $20. *(Editor's note: I just looked, and they have them for $3.49!)* The biggest difference between the Fluke and the cheapies is the resolution speed. The Fluke reacts almost as fast as an analog meter. Some of the cheaper units update the screen like only every half second."


Harbor Freight

"My favorites are Fluke and Triplett. I've had my Fluke 29 series 2 for over 20 years now. It's even survived a fire in the shop. The case is a bit scorched but it still works perfectly!"

"We use the Milwaukee digital multimeter 2217-20 at my work and I really like them; they are tough little buggers too. The white on black readout works well in the dark since the screen is backlit; also is clear as can be in the sun."


Milwaukee Tools

In conclusion, a lot of any tool choice depends on what you're using it for. Quickie, low resolution checks? High precision troubleshooting? Professional or hobby? The answers above reflect a wide range of needs and solutions. Doing your research — as this poster did — is always going to be best.

LiPoly Battery Fires

Many combat bots and, of course, many other R/C hobby devices use Lithium Polymer batteries due to their incredible energy density. This allows lots of power to be packed into a very small, lightweight package.

Unfortunately, these batteries can sometimes catch fire with spectacular results. There are lots of guidelines and opinions about how to design LiPoly batteries into your bot. What to do if they catch fire is not always quite so clear.

A recent conversation among builders and event organizers yielded some great data. All this information has been validated through Material Safety Data Sheets and the National Fire Protection Association's website.

It's pretty universally acknowledged that the only "official" fire fighting tool is a class D fire extinguisher, specially made for fires like lithium and magnesium. Most venues don't provide these, as the dry powder is quite messy and not gentle on carpets, drapes, other bots, etc.

CO_2 extinguishers, commonly available, at best do nothing. Many sources feel the CO_2 actually reacts with burning LiPoly batteries, causing more problems than they help.

Many EOs provide buckets of sand or salt. Burying a small battery in either of these is an approved method — per MSDS — for containing a battery fire. Use two metal buckets, both with sand/salt in them. A pair of heavy welding gloves and a pair of cutters should be stationed with the buckets. If safely possible, the gloves can be used to transfer the battery into one bucket, then the contents of the other dumped on top. The entire bucket then goes outside the venue, following a pre-planned and pre-briefed egress route.

A specific substance for fighting these fires is copper powder, but let's face it. No one is going to have that lying around!

Many folks feel that salt water can be used to safely put out a burning LiPoly. Many safety organizations disagree with this. A LiPoly that is damaged or needs to be disposed of for some other reason can be soaked in salt water for several days to neutralize the chemistry. But, dumping a burning lithium battery in water — even salt water — is not recommended.

An interesting input from one builder: "You can get a five gallon bucket of the class D powder at a fire rescue supply shop. This can be used in place of salt or sand, and is much cheaper than getting a fire extinguisher refilled. It also is a lot less messy if you can get the battery out of the bot."

All EOs agreed that it's a partnership between the venue, organizer, and builder to make sure everyone knows what to do in case of a battery fire, and to follow the pre-briefed plan. SV

EVENT REPORT: Franklin Institute 2010 – The Rise of the Melty Brains!

● by Pete Smith

The Franklin Institute Science Museum in Philadelphia and NERC (**www.nerc.us**) presented their fourth annual robot event on Saturday, October 9, '10.

There was a good turn-out in most weight classes, and this made for a very busy one-day event. Competitors started arriving around 7:00 a.m. and by 10:30 a.m., everyone was through safety and ready to fight.

Of particular interest to most competitors were the three brand new "Melty Brain" bots (**Figure 1**) brought by Team Ready to Rumble. Melty Brains is a nickname of a design of bot that has been around for years but had not achieved wide acceptance due to the complexity of the electronics and software involved (complex enough to make your brain melt!).

In the last year or so, however, a breakthrough by Rich Olson of Seattle has resulted in a small electronics board that combined all the essentials required in a small and cheap package. Problems still remained with the fragility of the brushed drive motor used and that's where Ready To Rumble stepped in by developing a brushless version of the code that removed that fragility and opened the door to a design that could soon become commonplace.

A Melty Brain spins like a Thwackbot, but by switching the drive motors on and off at exactly the right time each revolution, it can also create translational movement. The bot can thus be very simple and solid, and by spinning its whole mass very rapidly it becomes a formidable opponent.

The design is also easily scaled up or down for the various weight classes. An Ant, Beetle, and Hobbyweight were entered at Franklin.

The one lb Antweight Melty "Little Spinny Tortoise Thingy" did well but "Zergling" managed to knock it out in a semi-finals fight.

The three lb Beetleweight "Spinning Tortoise" did better, getting to the finals only to lose to tough wedge/vertical disk bot "Mr. Croup."

Only the 12 lb version "Double Trouble" failed to get a prize. It started up in the wrong mode at the beginning of its fight against "Surgical Strike" so did not spin up to full speed, and quickly failed to translate anymore. However, that fight resulted in what was probably the biggest hit of the competition (**Figure 2**) and caused considerable damage to the arena. A short circuit during repairs resulted in damage that knocked "Double Trouble" out of the event.

Meltys had certainly showed their potential. Work still needs to be done on the translating as the other bots could show more aggression by taking the initiative. While they certainly hit hard, they appear to hit themselves about as hard or even more so as they

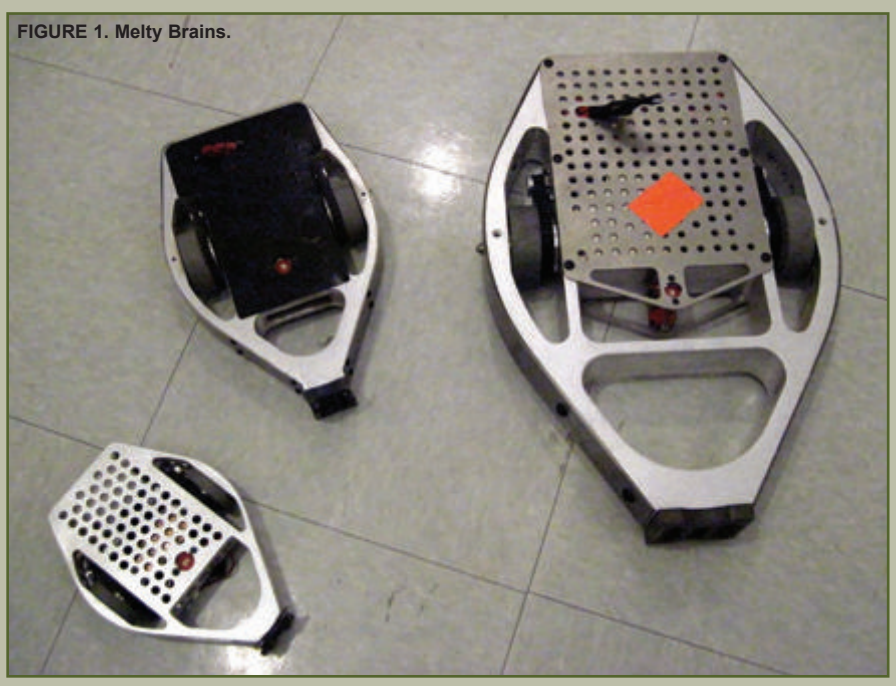


FIGURE 1. Melty Brains.

ricochet around the arena. Their main weakness appears to be that if you can get them stopped in a corner, they cannot easily get spun back up and escape. No doubt their builders are working on a solution to these issues just as their rivals are looking for a solution to this new threat.

The 30 lb Sportsman class grew again this year with new entries, most notably, the crusher "Lock-Jaw." It is seen here biting into "Upheaval" (**Figure 3**) to pierce his compressed gas supply line and forcing the Flipper to tap out.

The 30 lb Featherweights fought double round robin, and again full body spinner "Steel Shadow" showed promise but lacked the stability to make good use of its heavy shell. Drum spinner "Higgins" took advantage of this to win their fight (**Figure 4**).

Fights continued all day (without a break for lunch!) and even past the original end time for a tiring but exciting day for all involved. NERC's next big event is Motorama in February 2011. SV

Photographs by Brian Benson ***www.bensonpv.com*** *and Author.*

FIGURE 2. Double Trouble versus Surgical Strike.

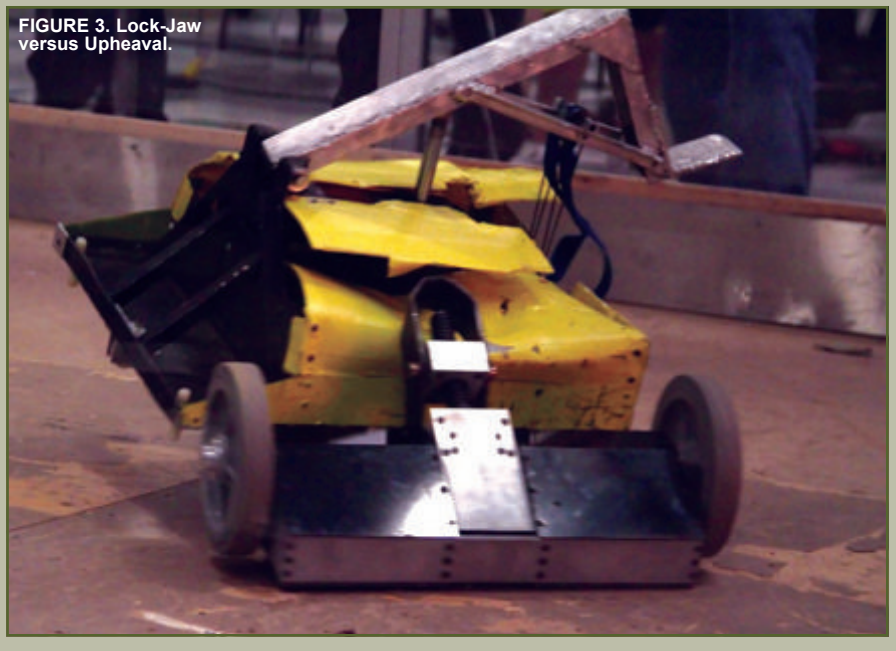


FIGURE 3. Lock-Jaw versus Upheaval.

FIGURE 4. Steel Shadow versus Higgins.

RESULTS

One lb Ants
1st – Szalor Service Shuffle Bot
2nd – Zergling
3rd – Little Spinny Tortoise Thingy

Three lb Beetles
1st – Mr. Croup
2nd – Spinning Tortoise
3rd – Torgr

12 lb Hobbyweights
1st – Scurrie
2nd – Surgical Strike
3rd – Flat Line

30 lb Sportsman
1st – Mangi
2nd – Lock-Jaw
3rd – Shish Kabot

30 lb Feathers
1st – Higgins
2nd – Steel Shadow
3rd – The Duke of Death

The Franklin Cup was awarded to Team "Ready to Rumble."

Lynxmotion

The Lynxmotion Servo Erector Set
Imagine it... Build it... Control it!

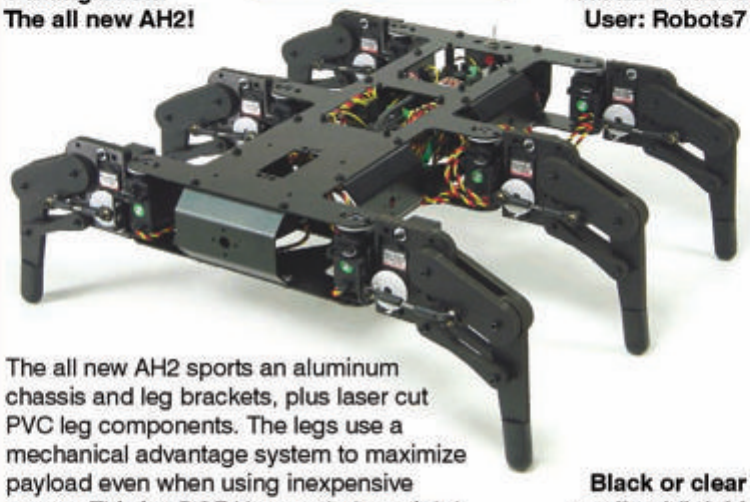
Biped Nick

Biped Pete

Featured Robot

Coming Soon!
The all new AH2!

Youtube videos
User: Robots7



The all new AH2 sports an aluminum chassis and leg brackets, plus laser cut PVC leg components. The legs use a mechanical advantage system to maximize payload even when using inexpensive servos. This is 2DOF Hexapod, done right!

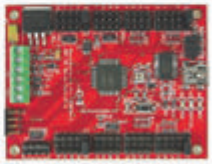
Black or clear anodized finish!

Biped Scout

Biped 209

With our popular Servo Erector Set you can easily build and control the robot of your dreams!

Our interchangeable aluminum brackets, hubs, and tubing make the ultimate in precision mechanical assemblies possible.



All New ARC-32 - $99.95
32 Channel Servo/Microcontroller.
USB Programming port.
32 bit Hardware based math.
Program in BASIC, C, or ASM
Servo and Logic power inputs.
Sony PS2 game controller port.

Bot Board II - $24.95
Carrier for Atom / Pro, BS2, etc.
Servo and Logic power inputs.
5vdc 250mA LDO Regulator.
Buffered Speaker.
Sony PS2 game controller port.
3 Push button / LED interface.

SSC-32 - $39.95
32 Channel Servo Controller.
Speed, Timed, or Group moves.
Servo and Logic power inputs.
5vdc 250mA LDO Regulator.
TTL or RS-232 Serial Comms.
No better SSC value anywhere!

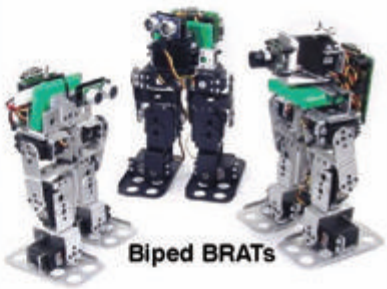
We also carry motors, wheels, hubs, batteries, chargers, servos, sensors, RC radios, pillow blocks, hardware, etc!

Visit our huge website to see our complete line of robots, electronics, and mechanical components.

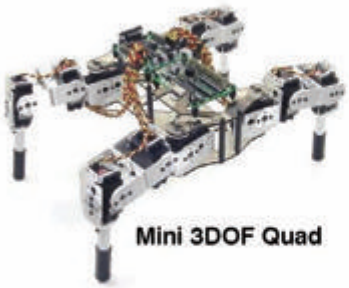
Walking Stick

CH3-R

T-Hex



Biped BRATs



Mini 3DOF Quad

Phoenix

Images represent a fraction of what can be made! **www.lynxmotion.com** **The SES now has over 200 unique components!**

Rate the eM8
Part 2

By Fred Eady

You can't build robot smarts if you can't speak the language. Or, in this case, speak the languages. We're about to continue our port of the eM8 assembler application to CCS C. Although this is a tutorial of sorts, we're also forming a picture of the eM8 hardware as we translate the assembler source code. When we ran out of paper last time, we had just finished porting the PIC16F882 definitions and SRAM allocations. At this point, we're ready to port some eM8 executable code.

Start

Assembler-based embedded programs don't have to house their code inside the braces of the C *main* function. However, from a porting standpoint, the assembler code behind the START label will be rounded up and placed within a pair of CCS C braces. We don't have to port the RESET code, but I thought it might be a good idea to show you how the assembler application kicks off following a microcontroller reset:

```
;-----------------------------------------
; Start Of Program after Reset
;-----------------------------------------

RESET      ORG      0x000     ; processor reset
                              ; vector
           GOTO     START     ; go to beginning of
                              ; program
```

The interrupt handler code lies between the RESET label and the START label in the original eM8 source. We'll deal with the interrupt handler port in due time. For now, let's dig into the oscillator setup:

```
START
;*******  Set Oscillator Factoty Correction
 BANKSEL      OSCTUNE  ; select bank 1
                       ; using mpasm macro
 MOVLW        0x00     ; set oscillator to
                       ; factory calibrated freq
 MOVWF        OSCTUNE
 BANKSEL      OSCCON
 BSF          OSCCON,IRCF0
                       ; set osc to 8M operation
 BANKSEL      STATUS   ; select bank 0 using mpasm
                       ; macro
 clrf         UF
 clrf         CF
```

We already know that the PIC16F882 is clocked from its internal oscillator. So, according to the PIC16F882 datasheet, the OSCTUNE register will reset with a value of 0x00 which does not apply any frequency correction to the default clock frequency of 4 MHz. At this point, there is no reason to adjust the instruction clock frequency. However, we do need to set up the internal oscillator to run at 8 MHz. As for setting the internal oscillator base frequency, CCS C provides a couple of ways for us to specify the instruction execution rate. The first method is found in the *em8.h* include file that is created by the CCS C PIC Wizard:

```
#use delay(clock=8000000)
```

The CCS C *#use delay* directive sets the necessary bits in the associated SFRs (Special Function Registers) to select an internal oscillator speed of 8 MHz. As you can see in the eM8 assembler source, bit IRCF0 in SFR OSCCON is set to complete the bit pattern necessary to initiate 8 MHz internal oscillator operation. Here's method number two which happens to be a CCS C built-in function that resides inside of the C *main* function:

```
setup_oscillator(OSC_8MHZ);
```

It doesn't hurt to specify the internal oscillator frequency more than once. However, the most recently encountered frequency statement sets the operating clock speed.

The eM8 application also clears the flags in this assembler code snippet and so will we. The original eM8 assembler source allocated an eight-bit SRAM location for each set of flags. We did the same but in a CCS C kind of way. The results of clearing the *UF* and *CF* flag structures are shown in **Screenshot 1**. The CCS C code that cleared the bits in **Screenshot 1** looks like this:

```
UF = 0;
CF = 0;
```

The eM8 assembler source makes provisions for Versions 1 and 2:

```
;_____________________________________________________
; Assembler Processing
;{
Ver    set     2
```

I/O Port Setup

From the looks of things, we have Version 2 eM8 hardware. So, instead of porting the Version 1 versus Version 2 decision trees, we'll get up on our donkeys and port for Version 2 only. The first time we have to make a version decision occurs when we set up PORTC. Meanwhile, we've got some PORTA and PORTB C coding to do:

```
;*******  Set IO and port ****
        banksel      TRISA
        movlw        0x0F
        movwf        TRISA   ; set port A for
                             ; OUTPUT
```

Porting the PORTA setup code is a walk in the park for CCS C:

```
set_tris_a(0x0F);
```

The same goes for the PORTB port. However, we've got an assembler macro to port, as well:

```
;_____________________________________________
        movlw        0x3D    ; io 8 output and
                             ; Int input
        movwf        TRISB   ; set RB0 for input
                             ; other io for input
        banksel      PORTB
        movf         PORTB,w
        OUT4_off             ; set lower IO off
```

Here's the CCS C PORTB setup port:

```
set_tris_b(0x3D);
```

The *OUT4_off* macro is coded as follows:

```
OUT4_off      macro
       banksel PAB
       bcf           PAB,4
       bcf           PAB,5
       bcf           PAB,7
       bcf           PAB,6
       PORTA_out
       Endm
```

PORTA_out is yet another macro that is called from the *OUT4_off* macro:

```
PORTA_out     macro
       movf   PAB,w
       movwf  PORTA
       endm
```

We're not here to second guess the intent of the assembler coder. We're here to port assembler code and gain an understanding of the eM8 hardware. So, instead of shortcutting the assembler mnemonics, we'll port line by line as best we can and try not to lose anything in the translation. Let's begin by transforming the *OUT4_off* macro to CCS C speak:

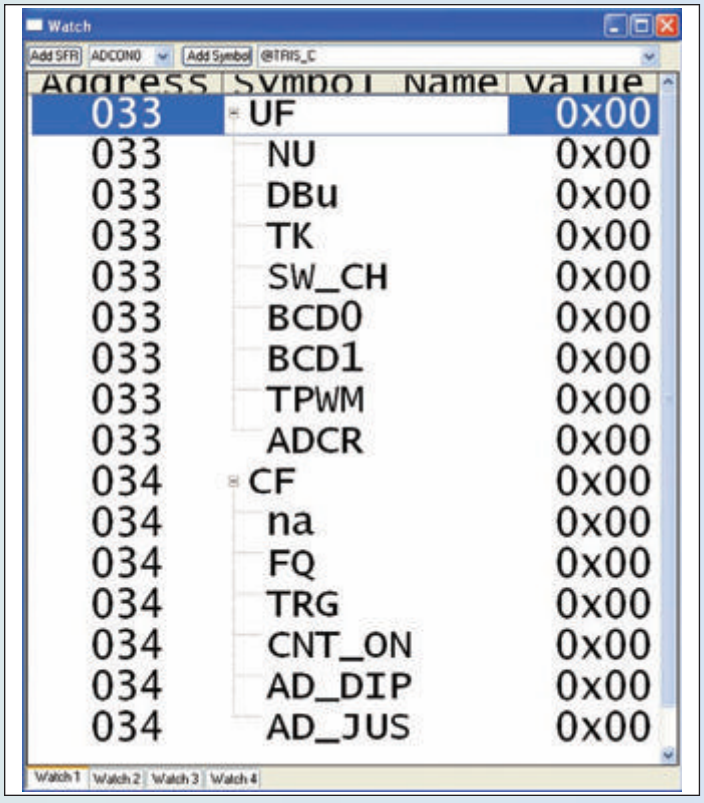
```
#define OUT4_off     bit_clear(PAB,4);     \
                     bit_clear(PAB,5);     \
                     bit_clear(PAB,7);     \
                     bit_clear(PAB,6);     \
                     output_a(PAB);
```

In this case, nothing will be lost in the translation if we usurp the *PORTA_out* macro with the CCS C built-in function *output_a*. There is an identical PORTB_out assembler macro which we will also replace with the CCS C built-in function *output_b*. I've placed our newly ported CCS C *OUT4_off* macro in a file we will call *eMate_MACRO.inc*. If we had chosen to pay attention to versions, all we would have to code is this:

```
#define Ver   2
```

However, we did not. So, let's move on and finish up the I/O port setup porting. We'll take the Version 2 track in the assembler code snippet that follows:

```
;_____________________________________________
        banksel      TRISC
if Ver == 1
```



SCREENSHOT 1. This shot is courtesy of a MPLAB Watch window and a Microchip PICkit3. The original flag names are preserved in both the UF and CF bit structures. The na is short for not available and is a place holder for three unused bits. From top to bottom, the bits run least significant bit to most significant bit.

```
        movlw         0x18
else
       movlw          0xD8

endif
        movwf         TRISC
        banksel       PORTC
        movf          PORTC,w   ; set IO for PORTC
```

I think you already have the CCS C equivalent in mind:

```
set_tris_c(0xD8);
```

The CCS C compiler will automatically set an I/O port pin for input or output depending on the C statement that is being executed against the I/O pin. For instance, the built-in *output_bit* function will force the compiler to set the targeted I/O bit as an output pin. Naturally, the *input_bit* function will cause the I/O pin of interest to become an input pin. *Fred Eady's First Rule of Embedded Computing* states that nothing is free in the embedded computing world. The price we pay for the compiler to automatically change the direction of I/O pins is time. If we choose to take care of I/O pin direction ourselves, time will be conserved. To take control of the I/O pin direction, we will add the following line of CCS C source:

```
#use fast_io(ALL)
```

Meanwhile, back at the ranch, the next assembler coding event simply clears the DPOINT byte:

```
if Ver == 1
       call           GETDIP   ; get dip switch
                               ; settings
else
       clrf           DPOINT
endif
```

That's a CCS C no brainer:

```
DPOINT = 0;
```

That does it for the porting of the I/O port initialization code. We're ready to move into PIC16F882 on-chip peripheral territory. Our path has been chosen for us.

Analog-to-Digital Setup

Much of the assembler code will translate to CCS C built-in functions. For instance, here's the analog-to-digital assembler code that selects the number of (A-to-D) channels:

```
;_____________________________________________________
       bankselANSELH ; set inputs AN0-3
       clrf   ANSELH ; adc analouge inputs
                     ; set up
        banksel    ANSEL
        movlw 0x0F
        movwf ANSEL
;_____ ADC _____
        call          AD_SETUP
```

The compiler has a built-in function that selects the (A-to-D) channels and the reference source. In addition, the PIC Wizard generates a built-in function that — in the case of initialization — turns the (A-to-D) peripheral off:

```
setup_adc_ports(sAN0|sAN1|sAN2|sAN3|VSS_VDD);
setup_adc(ADC_OFF);
```

Once the analog inputs are defined, a subroutine is called. A subroutine in C is called a function. With that, let's port our first assembler subroutine to a CCS C function. Here's the *AD_SETUP* subroutine assembler source:

```
AD_SETUP:  ;*******  Start A to D converter
       movlw          0x00
; set up ADC SFR to read ADC
       movwf          ANSELH
       banksel        ADCON0
       bcf            ADCON0,CHS0
; read dip switchs and select AN channel
       btfsc          DS_READ,0
       bsf            ADCON0,CHS0
       bcf            ADCON0,CHS1
       btfsc          DS_READ,1
       bsf            ADCON0,CHS1
       banksel        ADCON0
       movlw          0xC1
       movwf          ADCON0
       banksel        ADCON1
       clrf           ADCON1
       bsf            ADCON1,ADFM
; set for Right Just of ADC results
       bankselPIE1
       bsf            PIE1,ADIE
       banksel        0
       bsf            ADCON0,GO
       bsf            INTCON,PEIE
       bsf            UF,ADCR
       return
```

We've already taken care of ANSEL with the setup_adc_ports built-in function call. Although I believe the DS_READ code has to do with previous DIP switch reads, right now we really don't know why bits zero and one of DS_READ are being queried. So, let's not get up on our donkey and make any proclamations. Let's just do the porting:

```
void AD_SETUP(void)
{
       set_adc_channel(0);
       if(bit_test(DS_READ,0))
       {
              set_adc_channel(1);
       }
       if(bit_test(DS_READ,1))
       {
              set_adc_channel(2);
       }
       setup_adc(ADC_CLOCK_INTERNAL );

       enable_interrupts(INT_AD);
       read_adc(ADC_START_ONLY);
       UF.ADCR = 1;
}
```

The really cool thing about the CCS C built-in functions is that they are self-commenting. All we know right now is that DS_READ is the holding SRAM location for the DIP switch value. The DIP switch settings determine which portion of the educational program the eM8 runs. The *read_adc(ADC_START_ONLY)* built-in function call kicks off an (A-to-D) sample event but does not read the result.

That's Far Enough

We've split a bunch of assembler code logs. It's about time we start a C fire. My motto is, "If you can blink the LEDs, you can master the rest of the hardware." With that, I did a fast forward to what I thought would be the LED control code in the *eMate_MACRO.inc* file. Some commented out assembler code was the bait that attracted me to this particular file:

```
;LED1_on  MACRO
;       banksel PAB
;       bsf             PAB,4
;       PORTA_out
;       endm
;LED1_off  MACRO
;       banksel PAB
;       bcf             PAB,4
;       PORTA_out
;       endm
;LED2_on  MACRO
;       banksel PAB
;       bsf             PAB,5
;       PORTA_out
;       endm
;LED2_off  MACRO
;       banksel PAB
;       bcf             PAB,5
;       PORTA_out
;       endm
;LED3_on  MACRO
;       banksel PAB
;       bsf             PAB,7
;       PORTA_out
;       endm
;LED3_off  MACRO
;       banksel PAB
;       bcf             PAB,7
;       PORTA_out
;       endm
;
;LED4_on  MACRO
;       banksel PAB
;       bsf             PAB,6
;       PORTA_out
;       endm
;LED4_off  MACRO
;       banksel PAB
;       bcf             PAB,6
;       PORTA_out
;       endm
```

The commented LED control code was immediately followed by active code that replaced the LEDx wording with IOx. Here are a couple of IOx examples:

```
IO1_on        Macro
       banksel PAB
       bsf             PAB,4
       PORTA_out
       endm
IO1_off       Macro
       banksel PAB
       bcf             PAB,4
       PORTA_out
       Endm
```

Hmmmmm ... I also wondered why the macros were ordered 4-5-7-6 instead of 4-5-6-7. The answer lies in the way the PIC16F882 is pinned out. If you trace the PIC16F882 pinout from a PORTA perspective — from the least significant bit up — it is pinned like this: RA0, RA1, RA2, RA3, RA4, RA5, RA7, RA6. Remember, the eM8 is a teaching tool and the assembler code following the actual pinout is totally logical.

The only way to test our LED hunch is to write some C code and see if the LEDs follow. We already know that the least significant bits of PORTA are dedicated to the PIC16F882's (A-to-D) converter subsystem. So, our LED code is written to work against the most significant bits of PORTA:

```
int8 x;

do{
       for(x=0;x<16;++x)
       {
              output_a(x<<4);
              delay_ms(200);
       }
       }while(1);
```

The idea behind our blinky code is to form a binary counter using the variable x. The value of x is then shifted to fit into the upper four bits of PORTA which map to the eM8's four LEDs. The gamble worked and executing the C blinker code forced the LEDs into a regimented binary count sequence.

With the conquest of the eM8's LED Island, we now have a way to visually indicate locations in the code and states of operation. In other words, we have a debug indicator we can use to move further into understanding the inner workings of the eM8's hardware. We need only code some #define declarations to give our C-based application access to the LEDs. The LEDs can then be manipulated with the CCS C built-in I/O functions:

```
#define IO1   PIN_A4
#define IO2   PIN_A5
#define IO3   PIN_A7
#define IO4   PIN_A6
```

There is yet another debugging tool embedded within the eM8 hardware. The eM8 is sporting a brand new MCP2200 Serial-to-UART bridge. We've already discerned that the USB portal is not powering any portion of the eM8. However — USB power or not — there's really only one way to interface the MCP2200 to the PIC16F882's UART. Thus, we should be able to use the CCS C serial functions to use the MCP2200 as an I/O mouthpiece. The code is already in place inside the PIC Wizard-generated *eM8.h* file:

```
#use delay(clock=8000000)
#use
rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,
bits=8)
```

Sources

Queensland University of Technology; eM8
(Please contact Sam Wallace at: eM8@qut.edu.au for contact information and product details.)

CCS; CCS C Compiler
www.ccsinfo.com

While observing the eM8 running the original assembler application, I came to the conclusion that (A-to-D) converter channel zero was reading raw counts from the eM8's on-board MCP9701A temperature sensor. So, I chopped the original AD_SETUP code to only use the (A-to-D) converter channel zero:

```
void AD_SETUP(void)
{
       set_adc_channel(0);
       setup_adc(ADC_CLOCK_INTERNAL );
       enable_interrupts(INT_AD);
       read_adc(ADC_START_ONLY);
       UF.ADCR = 1;
}
```

To follow the flavor of the original assembler code, I populated the (A-to-D) converter interrupt handler which fires at the completion of each converter measurement event:

```
#int_AD
void  AD_isr(void)
{
       UF.ADCR = 0;
}
```

When flag bit UF.ADCR is cleared, a new (A-to-D) converter reading is available. Once the new converter temperature value has been captured, the UF.ADCR flag is set following the initiation of a new (A-to-D) converter conversion cycle. Let's place yet another bet on some code that will hopefully read the MCP9701A, compute the Celsius and Fahrenheit temperatures, print the temperatures via the MCP2200, and blink an LED to indicate activity. The whole shebang kicks off when a new temperature reading is captured:

```
int16 value;
float fvalueC,fvalueF;

   do{
          if(UF.ADCR == 0)
          {
              value = read_adc(ADC_READ_ONLY);
              read_adc(ADC_START_ONLY);
              UF.ADCR = 1;
              fvalueC = ((value * .00488) -
              .400)/ .0195;
              fvalueF = (fvalueC * 1.8) + 32;
              printf("Temp = %f C - %f
              F\r\n",fvalueC,fvalueF);
              }
          delay_ms(500);
          output_toggle(IO1);
       }while(1);
```

While the CCS C source code is relatively self-commenting, you do need to know where some of the numbers originated. The PIC16F882's (A-to-D) converter has 10 bits of resolution. The PIC16F882's converter reference is VDD which is jumpered for +5.0 volts on the eM8. Thus, each step of the PIC16F882's (A-to-D) converter is equivalent to 0.00488 volts. The MCP9701A's output voltage is scaled at 400 mV at 0° C with a temperature coefficient of 19.5 mV/°C. The floating point variable *fvalueC* is calculated using the MPC9701A's temperature characteristics according to this formula:

$$V_{OUT} = T_C * T_A + V_{0°C}$$

where:

T_A = Ambient Temperature

V_{OUT} = Sensor Output Voltage

$V_{0°C}$ = Sensor Output Voltage at 0°C

T_C = Temperature Coefficient

Once the Celsius temperature value is measured and calculated, the conversion to Fahrenheit is relatively simple. Both temperature values are displayed in the results you see in **Screenshot 2**. Thus far, we've succeeded in figuring out how to blink the eM8's bank of LEDs, read the eM8's MCP9701A temperature sensor using the PIC16F882's (A-to-D) converter, and display the temperature in Celsius and Fahrenheit. We've still got the seven-segment display hardware to figure out and code to port to make them useful.

Light 'Em Up

Rather than take the original assembler source and work it line by line, let's transition to a modular view of the code. I think we've done enough assembler-CCS C porting for you to be comfortable with the practice. I've surveyed the code and have formulated some theories as to how to control the bank of seven-segment LEDs. Most of my theories are based on the following segment of assembler code:

```
;*****   Display Bit Def
SO0     equ     0x0     ; BCD bit 0   RC0
SO1     equ     0x1     ; BCD bit 1   RC1
SO2     equ     0x2     ; BCD bit 2   RC2
SO3     equ     0x5     ; BCD bit 3   RC5
SO_SEL  equ     0x1     ; RB1 Control for DIP or
                        ; display 1 = Display
```

Basically, I believe the eM8 backbone is a four-bit bus. In addition to the SOx definitions, the four-bit I/O architecture of the MC14042, MC14028, MC14043, and MC14511 ICs also point to a four-bit bus. These comments also provide proof of the existence of a four-bit backbone bus:

```
;       SelH data to U9 4 bit data
;       SelL data to U18 select functions
```

The eM8's DIP switch and seven-segment display share the four-bit bus. Access to the four-bit bus is controlled by

Fred Eady can be reached via email at fred@edtp.com.

the SO_SEL output pin. SO_SEL is mentioned in the DIP switch read code and the use of SO_SEL corresponds with the SelH and SelL usage. For instance, a DIP switch read will first take the SO_SEL line logically low, then issue a 0x06 followed by a 0x07 or 0x04 to the SOx bus. The DIP switch is partitioned as a pair of four-bit switches with 0x07 selecting the upper bank and 0x04 selecting the lower bank. That means that 0x06 is a clear command of some type. The whole matter of writing to a seven-segment module is contained within this code:

```
Call          SelL
SelCL         0x05    ; set latch for Data clock
SelCL         0x0F    ; enable clock
SelCL         0x06    ; clear latch  **BCD Data
                      ; Latched to 4511 chip
SelCL         0x0F
SelC          o_gb0   ; send select to latch
Call          SelH    ; latch data to select
```

From the code, it looks like sending 0x05 is our portal to the seven-segment LED modules and placing 0x06 on the four-bit bus with SO_SEL logically low clears a latch. Since we're clearing latches, that means that U9 and U18 are of type MC14042. Now all we have to do is figure out how to manipulate and feed the latches using C. Let's begin by applying some aliases on the SO pins, RB1, SelH, and SelL:

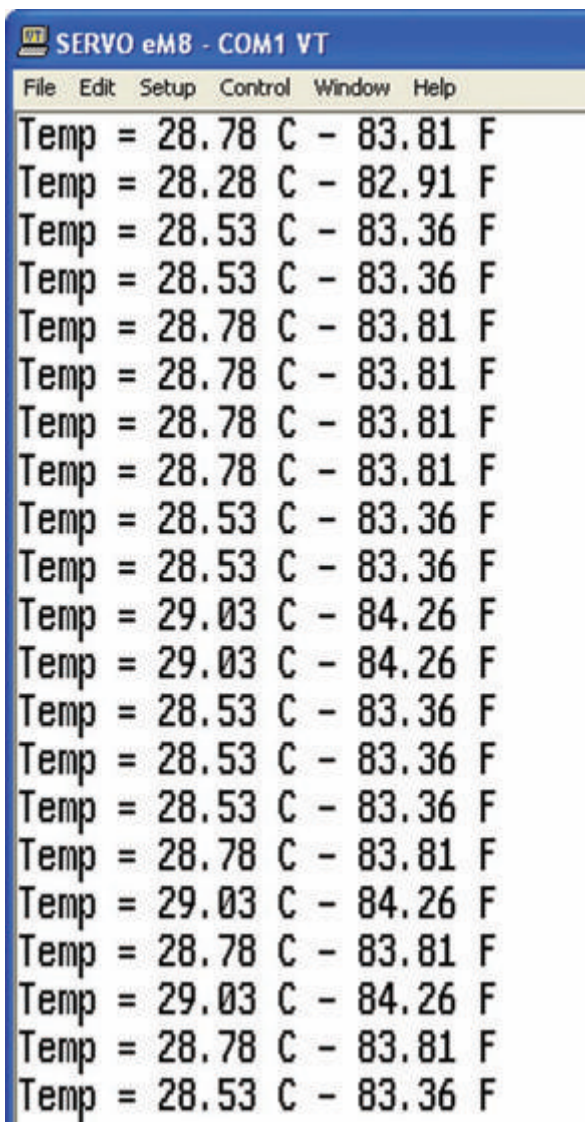
```
#define SO0           0x00
#define S01           0x01
#define S02           0x02
#define S03           0x05
#define SO_SEL        PIN_B1
#define DISPLAY       0x01
#define DIPSW         0x00
```

Judging from the electronic hardware driving the LED modules, I'm positive that the seven-segment display modules are multiplexed. That implies that the data to be displayed is loaded and latched just before the display module is activated. After a short period of time, data for the next seven-segment module is latched and displayed in the same manner as the display module before it. This process of latch and display moves from display module to display module in a never-ending loop. Here's our first throw at it:

```
do{
    LOAD_LEDS(0x01,0x00);
    delay_ms(1);
    LOAD_LEDS(0x02,0x01);
    delay_ms(1);
    LOAD_LEDS(0x03,0x02);
    delay_ms(1);
    LOAD_LEDS(0x04,0x03);
    delay_ms(1);
}while(1);
```

As you can see, the display code is nothing more than an endless *do-while* loop. The seeds that make the fruit look like this:

```
void LOAD_LEDS(int8 digit, int8 data)
{
  output_bit(SO_SEL,0);      //select DIP switch
                             //latch
  output_c(0x06);            //reset the DIP
                             //switch latch
  output_c(0x05);            //enable the MC14511
                             //for data
  output_c(0x0F);            //all zeros out from
                             //selected latch

  output_bit(SO_SEL,1);      //select DISPLAY
                             //latch
  PCB = data & 0x0F;         //clear upper nibble
                             //of data to display
  if(bit_test(PCB,3))        //adjust bit 3 of
                             //data
  {
       bit_clear(PCB,3);     //bit 3 was set
       bit_set(PCB,5);       //bit 5 is now set
                             //for bit 3
  }
  output_c(PCB);             //output data to
                             //DISPLAY latch
  output_bit(SO_SEL,0);      //latch DISPLAY
                             //data/select DIP
```



SCREENSHOT 2. I used my index finger and a can of compressed air to jog the MCP9701A's atmosphere.

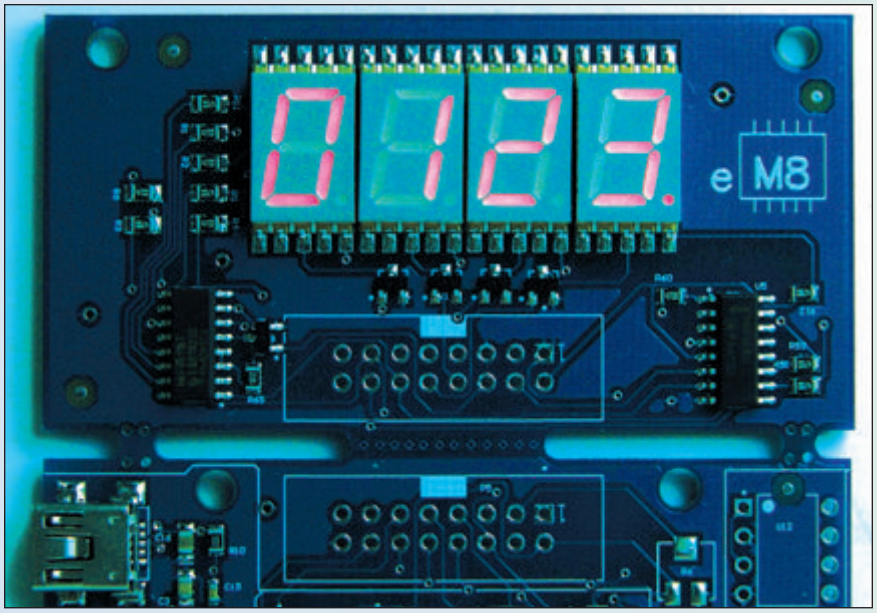


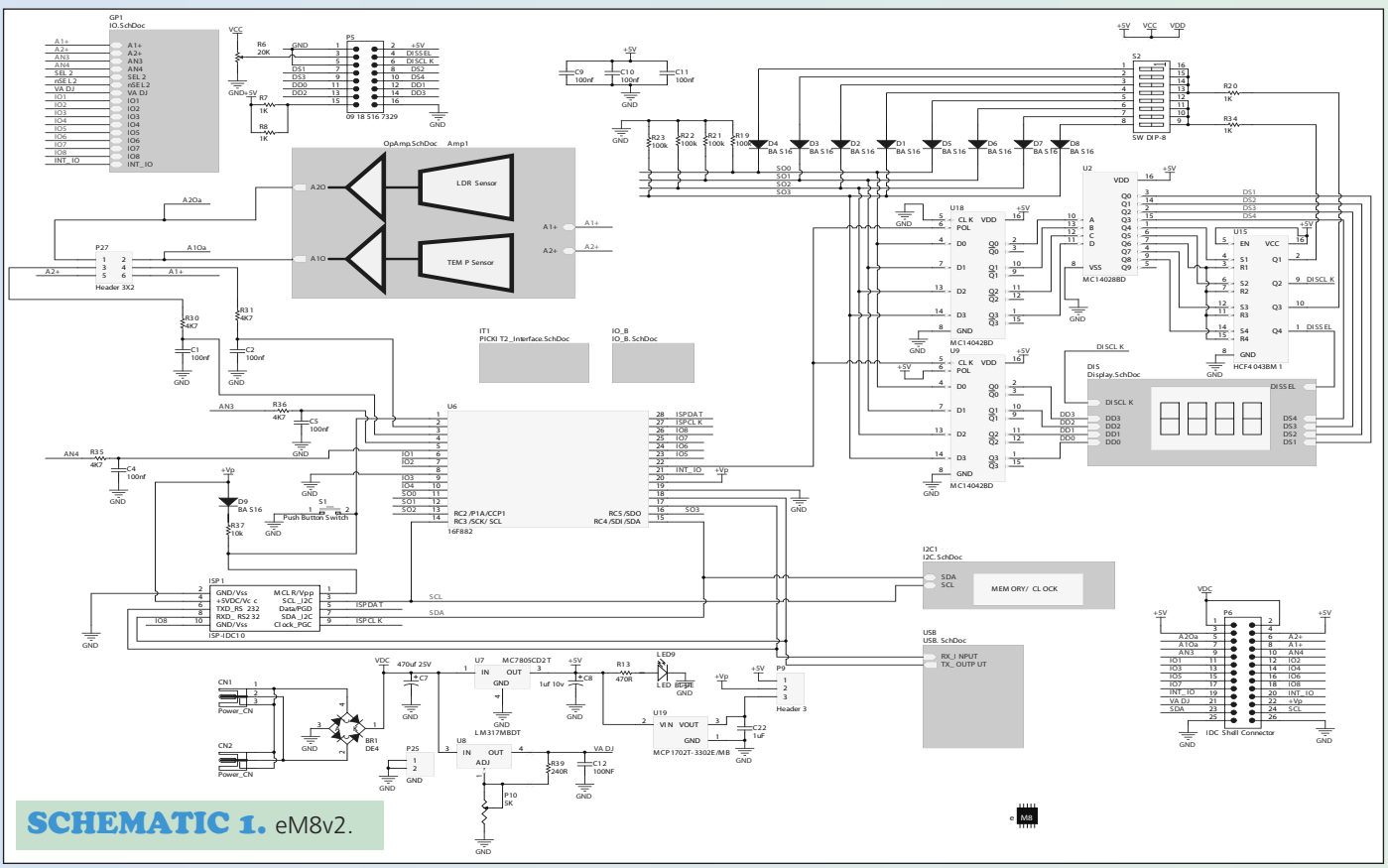
PHOTO 1. As far as the basics go, not much has been left to chance with the eM8. This board is built on the rock that supports microcontrollers and everything electronic.

```
                output_c(0x02);
//module 3
        break;
        case 4:
                output_c(0x03);
//module 4
        break;
        }

}
```

Funny ... The numbers 084 followed by a blank display module appeared. Hmmmm ... So, I changed the code to display all zeros and I got 0000 in the displays. The next logical step was to code for 1111. The result was 8888. Coding for 2222 gave me 4444, and 3333 produced blank displays. Okay. The MC14511 will only display 0-9 and blank above that. So, let's look at all of the combinations and results in binary:

```
0b00000001 becomes 0b00001000
0b00000010 becomes 0b00000100
0b00000011 becomes blank display
0b00000100 becomes 0b00000010
0b00000101 becomes blank display
0b00000110 becomes 0b00000110
```

```
                        //latch

switch(digit)           //choose 7-segment
                        //module
{
      case 1:
            output_c(0x00);       //module 1
      break;
      case 2:
            output_c(0x01);       //module 2
      break;
      case 3:
```



SCHEMATIC 1. eM8v2.

```
0b00000111 becomes blank display
0b00001000 becomes 0b00000001
0b00001001 becomes 0b00001001
```

Do you see the problem? The bit order is inverted. Only 6 and 9 invert to themselves. Inverting 0b00000011 results in 0b00001100. Plugging in 0x0C for 0x03 did, in fact, produce a 3333 display. The original assembler code used to "correct" the bit order of the data goes like this:

```
SelCC  macro          InCor
       clrf           seldat
       btfsc          InCor,0
       bsf            seldat,3
       btfsc          InCor,1
       bsf            seldat,2
       btfsc          InCor,2
       bsf            seldat,1
       btfsc          InCor,3
       bsf            seldat,0
       call           SelCCCL
       endm
```

Our equivalent CCS C "correction" code is written this way:

```
    if(bit_test(data,0))
        display_data += 0x08;
    if(bit_test(data,1))
        display_data += 0x04;
    if(bit_test(data,2))
        display_data += 0x02;
    if(bit_test(data,3))
        display_data += 0x01;
```

With the addition of the bit order correction code, our seven-segment LED is fully operational and has taken this form:

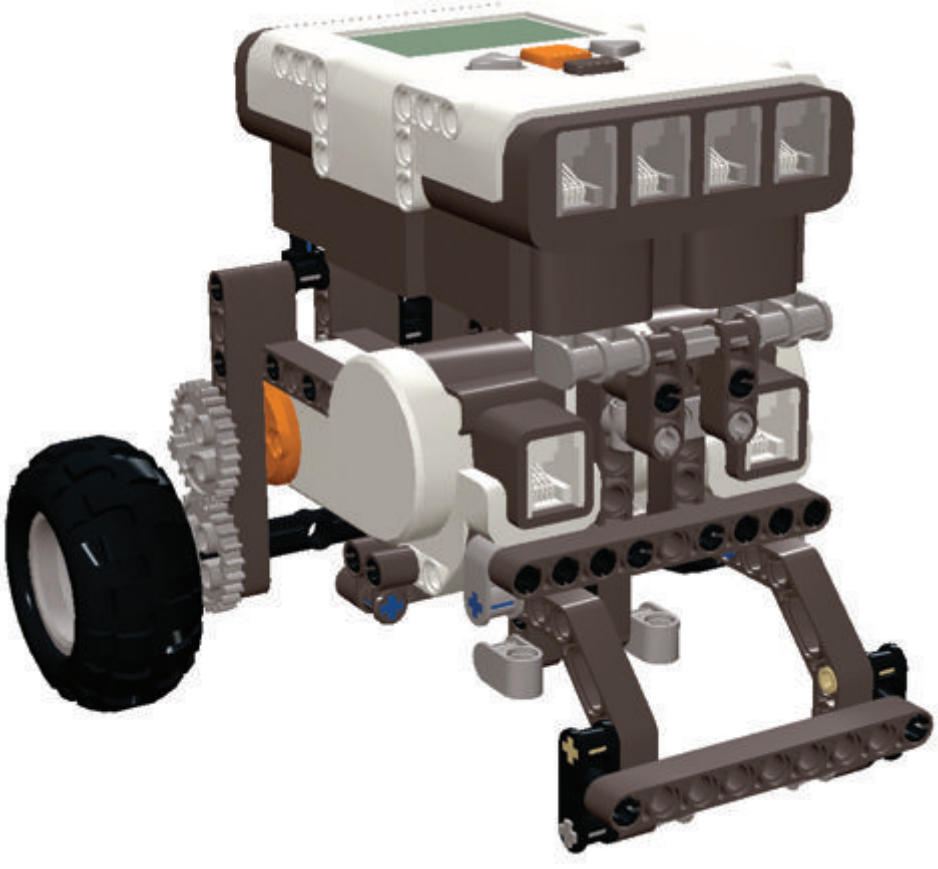
```
void LOAD_LEDS(int8 digit,
int8 data)
{
 output_bit(SO_SEL,0);
//select DIP switch latch
 output_c(0x06);
//reset the DIP
//switch latch
 output_c(0x05);
//enable the MC14511
//for data
 output_c(0x0F);
//all zeros out from
//selected latch

 output_bit(SO_SEL,1);
//select DISPLAY
//latch
 display_data = 0x00;
//clear display data byte
 if(bit_test(data,0))
       display_data += 0x08;
 if(bit_test(data,1))
       display_data += 0x04;
 if(bit_test(data,2))
       display_data += 0x02;
 if(bit_test(data,3))
       display_data += 0x01;

 PCB = display_data & 0x0F;
//clear upper nibble
//of data to display
if(bit_test(PCB,3))          //adjust bit 3 of
//data
 {
       bit_clear(PCB,3);     //bit 3 was set
       bit_set(PCB,5);       //bit 5 is now set
                             //for bit 3
 }
 output_c(PCB);              //output data to
                             //DISPLAY latch
 output_bit(SO_SEL,DIPSW);   //latch DISPLAY
                             //data/select DIP
                             //latch

 switch(digit)               //choose 7-segment
                             //module
       {
       case 1:
              output_c(0x00);      //module 1
       break;
       case 2:
              output_c(0x01);      //module 2
       break;
       case 3:
              output_c(0x02);      //module 3
       break;
       case 4:
              output_c(0x03);      //module 4
       break;
 }
}
```

As we exposed the inner workings of the eM8, I've shown you everything I can snag in a screenshot or put under the lens of my Canon. The electronic version of the 123's is the subject of **Photo 1**.

You Get a Schematic

We've explored the eM8 with a blind eye. The good news is that when you purchase your own eM8, it comes with a schematic. SV

The NXT Big Thing #5



Light-Hearted

By Greg Intermaggio

Last time, we learned how to make a robot that can follow a line accurately by using two light sensors. This month, we'll be learning about a very important programming concept: dynamic variables, and we'll be doing it by making Eddie seek light!

This time, we'll take a look at dynamic variables and data wires:

- Just what are dynamic variables?
- Why are they so darned important?

After that, we'll build a new attachment for Eddie, then get to programming!

Just what are dynamic variables?

We mentioned dynamic variables last month, but didn't go into much depth beyond describing the different types. We know that a variable can be binary, boolean, or string but what does it all mean? Well, it turns out that using *dynamic* or *changing* variables can be an incredibly powerful tool in our programming arsenal.

Consider the following: Let's say we wanted Eddie to move faster if he saw a bright light, or slower if he didn't see any light. There is a variable for motor power (speed) that is set to 75 by default in the NXT software. This number means that 75% of the maximum motor power will be given to the designated motor, and the number can be anywhere between 0 (no power) and 100 (full power). You can adjust it manually, or you can get fancy and use dynamic variables to adjust the motor every time the light value changes.

This is even cooler because the light sensor's variable is also a percentage (0-100). So, the brighter the light, the higher the variable which means that if we correspond that variable to motor power, we don't have to do any math! We'll learn more about how to do this once we get to programming.

Why are they so darned important?

Think of the possibilities! Dynamic variables can be used in just about everything — from calculating interest rates on bank accounts based on how much money the bank has in reserves, to determining the average amount of toxic chemicals in the air after an explosion before sending humans into the rubble.

Dynamic variables are one of the fundamental building blocks of advanced programming, and today we'll be getting ourselves very comfortable with them.

First, let's build a new attachment for Eddie that will aim his light sensors outward so he can see the light around him, instead of under him.

Building Instructions Light Seeker Attachment

1. Start with a single 4 x 6 angular studless beam.

2. Add two black friction pegs and two blue hybrid pegs as indicated.

3. Attach the light sensor to the shorter end.

4. Replicate the attachment in reverse — this will attach to the other side of the robot.

5. This is the finished product. Plug the sensor on Eddie's left into port 1 and the one on the right into port 2.

Testing a Dynamic Variable

We've built the attachment. Now, let's create a program that uses a dynamic variable. This program will make Eddie move forward faster if he sees a bright light, slower if he sees little light, and stop if he sees no light.

Let's take a moment to think about what we just did:

- We connected the "intensity" variable output of the light sensor to the "power" variable input of the motor block.
- We've set the motor movement to backwards, which

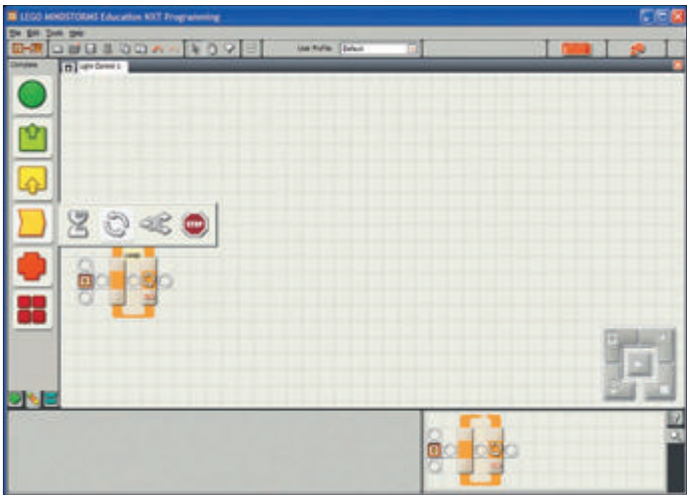


Figure 1. Start with a loop.

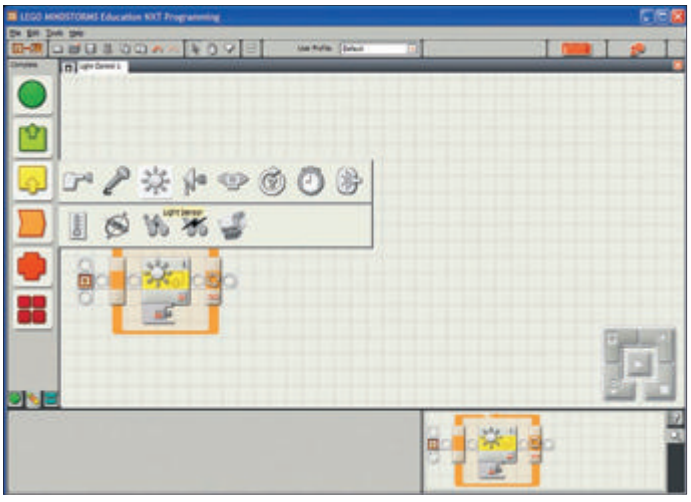


Figure 2. Add a light sensor check. Make sure you find this under "Sensor" in the complete palette — DO NOT use the "Wait" block. Also, be sure to uncheck "Generate Light" in the bottom panel and select port 1.

(because of Eddie's gear train) will make him move forward.

- We looped the program, meaning it will repeat the actions indefinitely.
- We know that the light sensor outputs a value between 0 and 100, with 0 being no light and 100 being very bright.
- Similarly, we understand that motor power is a percentage (meaning it will be between 0 and 100%).

Go ahead and test your program in a decently-lit room. If all goes well, Eddie should roll forward and slow down if you cover his left light sensor. You'll notice that there's a gradient of speed as you move your hand closer and closer. This is because less and less light is getting to Eddie's sensor, meaning he's supplying less and less power to the motors.

What if we want to make Eddie move faster when it gets dark, and slower when it's light? Well, it's actually fairly simple, and only requires one more block.

We'll need to take our input value (the light intensity) and apply a simple formula to reverse the values. First, let's consider the numbers, and make an in/out table to help us find the formula.

What we want here (see **table** here) is an input intensity of 0 leading to a motor power output of 100, and vice versa.

IN	OUT
0	100
100	0
50	50
25	75
75	25

Note that the IN + OUT values always equal a total of 100 — never more and never less. Knowing that, let's take a look at the first two rows

Figure 3. Drag in a move block. Set the direction to backwards (which will make Eddie move forward) and duration to unlimited. Finally, click the tab indicated to expand the "data hub" of the move action.

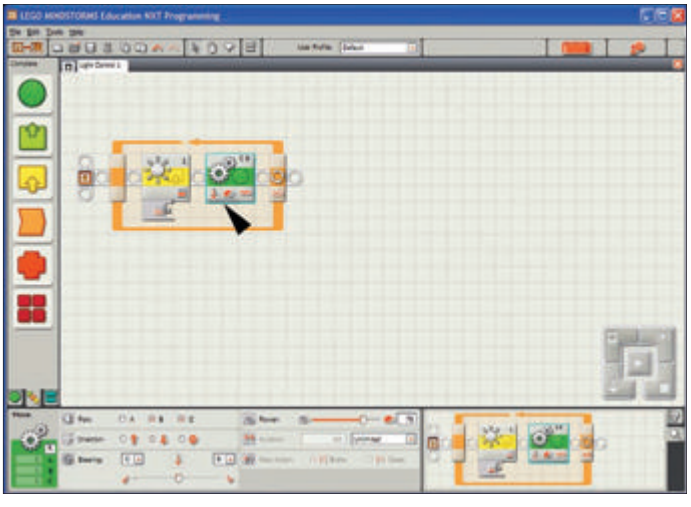
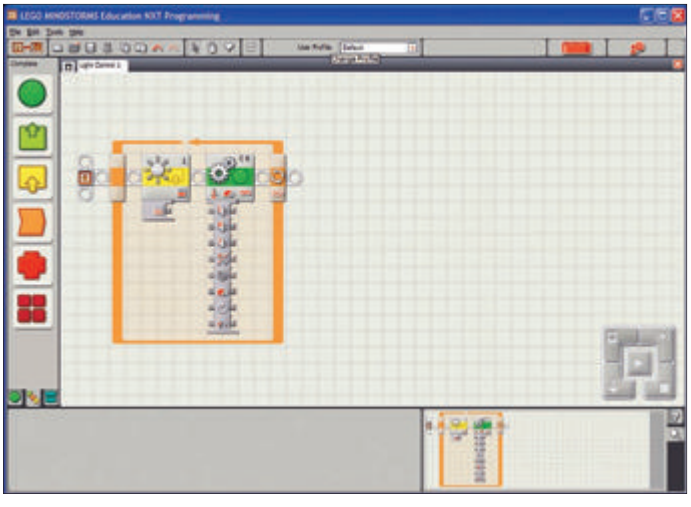


Figure 4. Here is the move action with the data hub expanded. Hover your mouse over the icons to see what each one is. Each of these are dynamic variables. They can be output (steering, for instance, would return a value between -100 and 100 corresponding to the direction of the robot, which could be used for other calculations) or they can be input, which we're about to do with motor power (we'll tell the motors how much power to use, based on another variable).



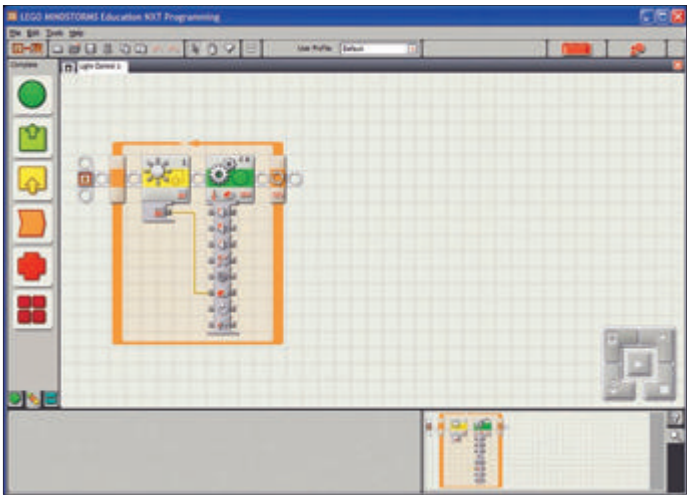


Figure 5. Click the "Intensity" port on the light sensor block, then the "Power" port on the motor block to connect the two with a "data wire."

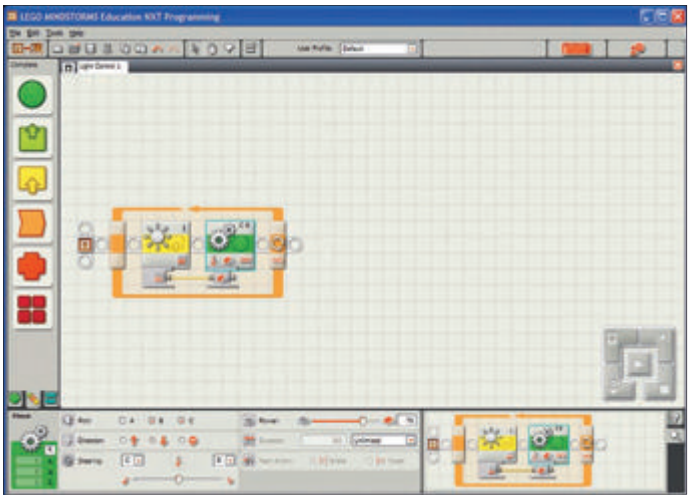


Figure 6. Click the tab under the move action again to collapse the unused data ports.

of values, and try to find the formula we'd need to make them a reality.

Start by considering the first row of values and asking yourself "What would I need to do to the number 0 to change it to the number 100?" The first possibility is obvious: You could add 100 to 0 to make it equal 100.

(0) + 100 = 100
or
(IN) + 100 = OUT

Let's try that with the second set of values:

100 + 100 =/= 0

So, that formula is a no-go. We could also try the opposite operation: subtraction. How could we use the number 0 in a subtraction operation to equal the number 100? We can't subtract a positive number from 0 because that would only make a negative output. We could, however, subtract 0 from the number 100 to equal 100.

100 - (0) = 100
or
100 - (IN) = OUT

Let's try that with the second row of values:

100 - (100) = 0

Looks good so far! Let's just try it with the other three rows:

100 - (50) = 50
100 - (25) = 75
100 - (75) = 25

Success! Our formula works! Now we just need to implement it in the program. We do this by using the "Math" block.

Figure 7. Start by clicking the data wire to select it, then pressing delete to delete it.

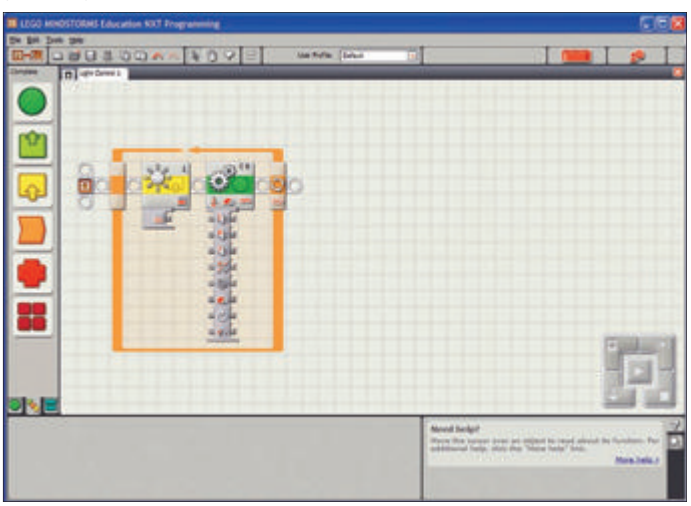
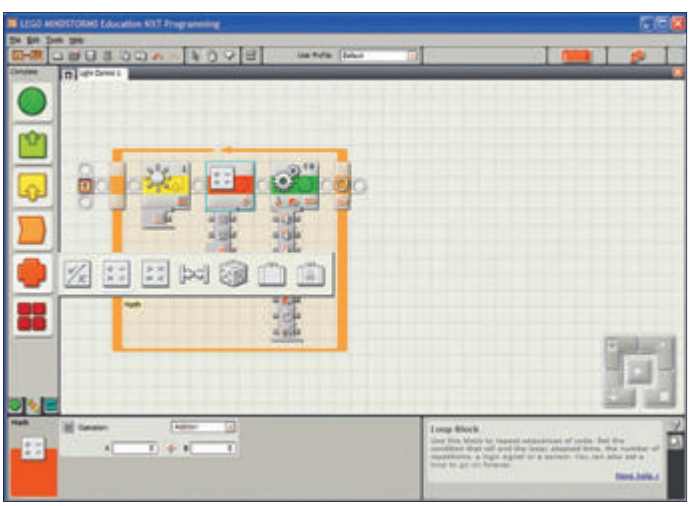


Figure 8. Find the "Math" block in the "Data" menu on the right and drag it *between* the light sensor and motor blocks.



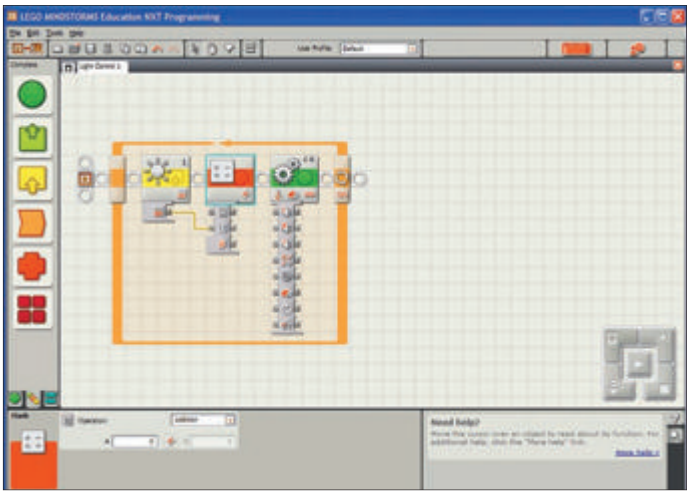


Figure 9. Connect the light sensor intensity data port to the "B" port on the math block.

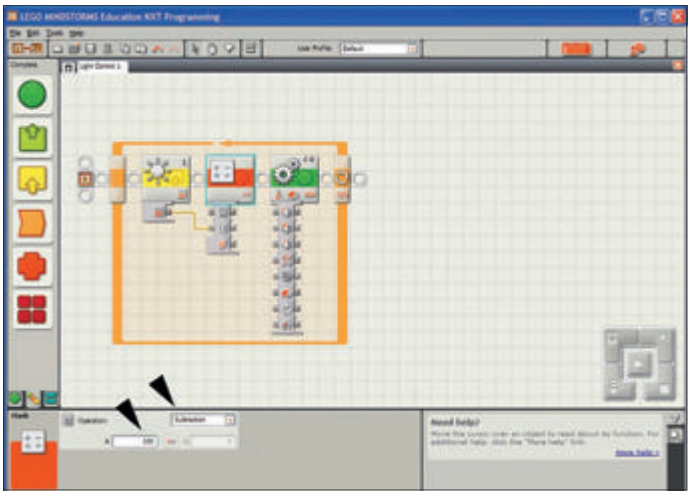


Figure 10. Select the math block, change the operation to "Subtraction," and type in "100" for "A."

Download your program to Eddie, and you're ready to rock!

Making Eddie Follow Light

Now that we understand how dynamic variables work, it's time to get on to the main challenge for this installment: Make Eddie follow a light!

First, we need to learn about steering. The steering data port on the move block accepts any numeric value between -100 and 100. An input of -100 will make Eddie turn left at full speed, while an input of 100 will make Eddie turn hard right. We'll need to do some math to find our formula to turn towards and approach light.

We have two light sensors, each of which can detect light intensity values between 0 and 100. If the light sensor on the left sees light (a higher intensity value), we want Eddie to turn left. Similarly, if the light sensor on the right sees light, we want Eddie to turn right. Take a look at the **table** here.

LEFT	RIGHT	STEER
100	100	0
0	0	0
100	0	-100
0	75	100
50	100	50
100	50	-50

This formula might seem more obvious than the last one, but in case it doesn't, let's consider the following:

- When no light is seen (both inputs are 0), the output is also 0. This suggests that there are no terms in the formula aside from the variables.
- When full light is seen (both inputs are 100), the output is still 0. This suggests that there is subtraction involved.
- Since we know that the equation likely only has two terms (LEFT and RIGHT) and we know that there is subtraction involved, the answer then becomes clear when you look at the third row of values:

RIGHT - LEFT = STEER

Unfortunately, we've forgotten that Eddie's gear train reverses the direction of his turns! Therefore, the *real* formula

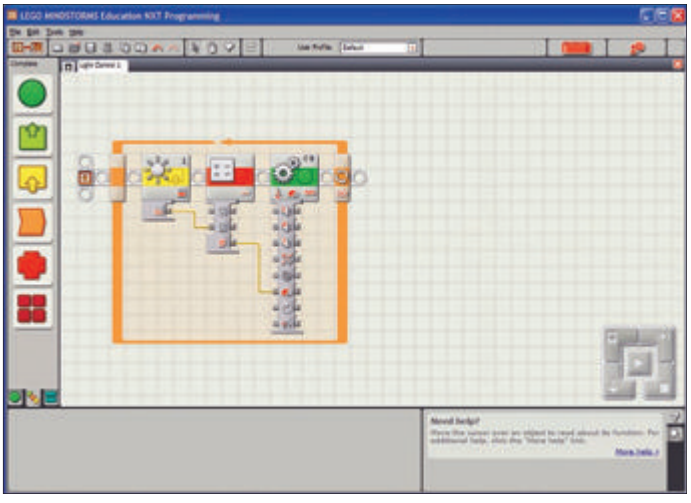


Figure 11. Finally, attach a data wire from the "#" output port on the math block to the power level input port on the motor block.

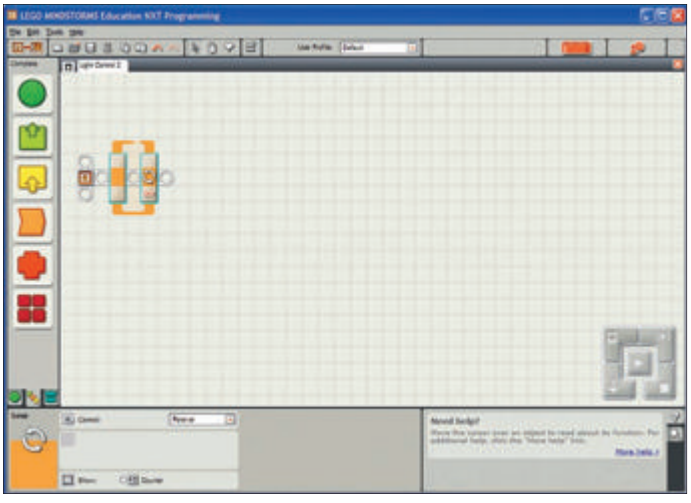


Figure 12. Start with a new program and add a loop.

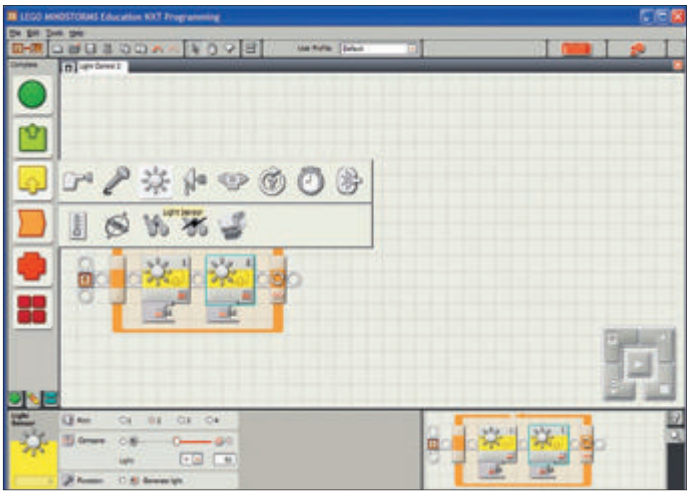


Figure 13. Drag in two light sensor blocks and select ports 1 and 2, respectively. Be sure to uncheck "Generate Light."

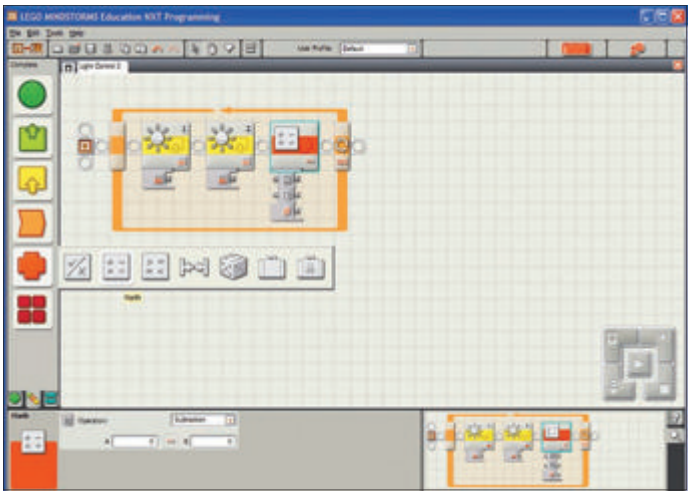


Figure 14. Find the math block and select "Subtraction."

we're looking for is:

LEFT - RIGHT = STEER

Close enough, right?

With that in mind, let's write the program.

Test the Program

Eddie should turn away from your hand if you cover one of his light sensors, or turn towards a flashlight if you shine it at him.

Experiment!

Now that we've gotten Eddie approaching the light, here are some fun experiments to try:

- Try different power levels to adjust Eddie's speed. Find your favorite!
- Change Eddie's gear train to further control his speed/torque.
- Try programming Eddie to respond more extremely to smaller changes in light (hint: multiplication).
- Try writing a similar program that uses the light sensors to control the motors independently. In other words, light sensor 1 controls the left motor (B) and light sensor 2 controls the right motor (C).

Wrapping Up

In this edition, we dove into dynamic variables and data wires. We programmed Eddie to react to light, then to follow it. Along the way, we learned about determining formulas based on input/output values.

In the January edition, we'll be making Eddie even cooler in an article you don't want to miss! Stay tuned! SV

Figure 15. Wire the sensors to the subtraction block as indicated. Note: Light sensor 2 should connect to port 1 on the subtraction block; light sensor 1 should connect to port 2.

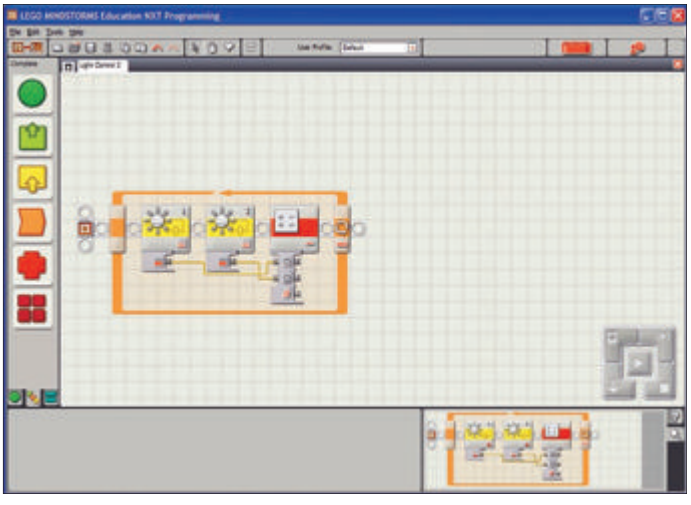
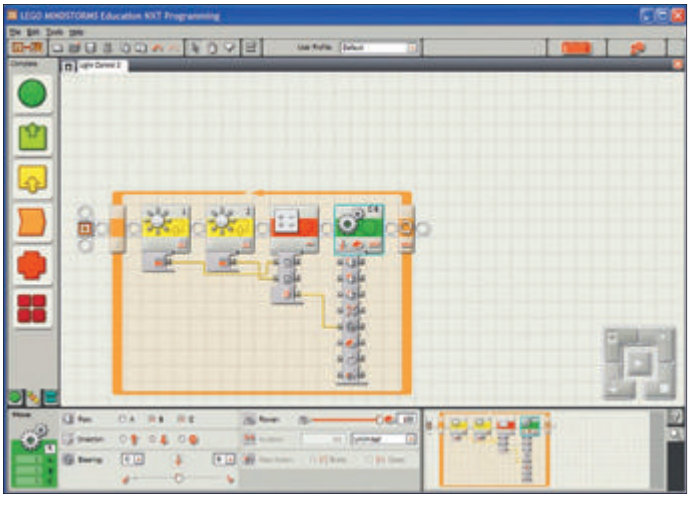
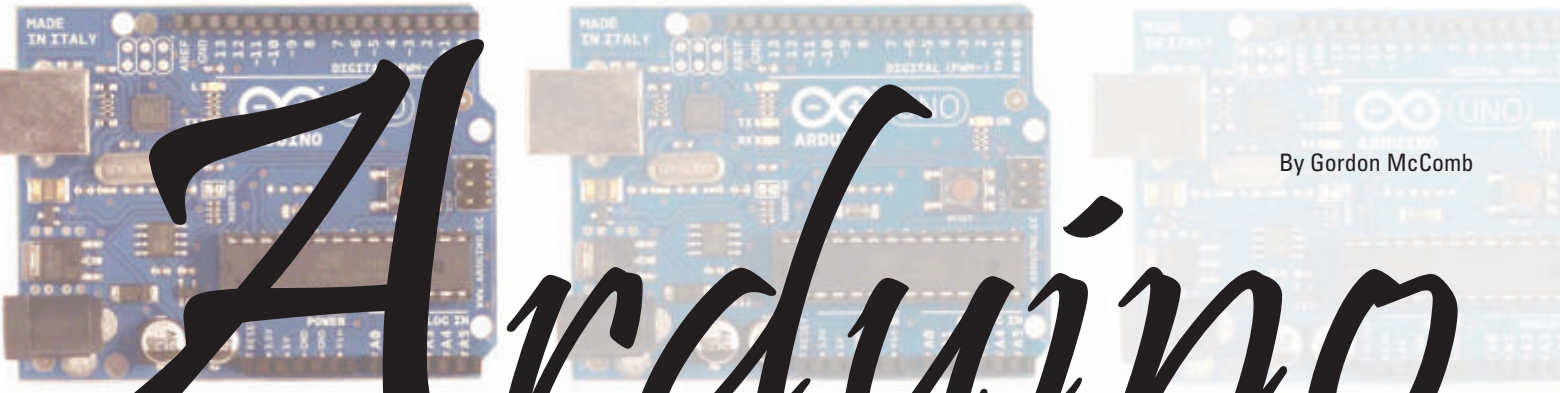


Figure 16. Finally, add a motor block. Set the direction to backwards, power to 100%, and duration to unlimited. Then, connect a data wire from the subtraction block output to the steering input on the motor block and you're done!



Making Robots With The Arduino

Part 2



By Gordon McComb

The ArdBot is a low-cost, 7" diameter servo-driven robot base, ready for expansion. It's called ArdBot because it's based on the popular and inexpensive Arduino microcontroller board. The ArdBot costs under $80 to build; even less if you already have some of the components, like the breadboard, jumper wires, and battery holder.

In the last installment, we introduced the ArdBot and its central Arduino brain. This month, we'll continue the discussion with full construction plans for the ArdBot. I built the reference design using 1/4" expanded PVC plastic, but you can use wood, acrylic, foam board, picture frame mat, or most anything else that is rigid enough for the components.

ArdBot Basic Design

The ArdBot uses two "decks" for mounting a pair of servo motors, batteries, microcontroller, small prototyping board, and other components you'd like to experiment with. The bottom deck is basically a 7" diameter circle with cutouts for the wheels. The top deck is the same 7" diameter circle with the side lobes cut off.

The decks are separated by a set of four 1-3/4" long standoffs. The actual length of the standoffs is not really important. You can make them shorter or longer — 1-1/2" is the practical minimum and 3" the maximum.

While it's a bit more challenging to cut circles to make a robot base, it's the best overall shape for navigating tight places like mazes or the corner of a living room. The concept of the ArdBot is flexibility, however. There's no reason your version must be circular. You can make a square bot if you'd like, or cut off the corners of the square to make an octagon.

If you don't want to construct the mechanical pieces of the ArdBot at all, you can get them precut with all the hardware; see the **Sources** box. ArdBot is designed for expandability. If the twin decks do not provide enough space for all your experiments, you can add more decks. I don't recommend any more than three decks total, as any more may pose a weight problem for the drive system.

The brain of the ArdBot is an Arduino Uno — the latest of the all-in-one core designs of the Arduino. If you already own an earlier version of the board — a Diecimila or Duemilanove — those will work, too. The only requirement is that you have version 0017 or later of the Arduino programming environment. The ArdBot project was created and tested using version 0019 — the latest as of this writing. Complementing the Arduino microcontroller board is a mini solderless breadboard. It has 170 tie points —

In preparing Part 1 of this series, I made a last-minute change to include the new Ardunio board that's just been released. Only I got the name wrong — in several places in the article, I referred to the new board as the Duo. The correct name for the board is the Uno.

enough for the basic experiments we'll be doing in this series of articles. Don't let the small size of the breadboard limit you. The ArdBot is large enough for bigger breadboards, even multiple boards, should you need them. You might want to start with the mini breadboard, then as you use the ArdBot for further experiments you can add more prototyping space.

About the Servo Drive

The ArdBot uses differential steering where the base is propelled by two motors and wheels on opposite sides. To keep costs down and minimize construction complexity, the robot uses a pair of skids placed in the front and rear to provide balance. With this arrangement, the ArdBot is able to move forward and back, turn left and right, and spin in place. The skids are smooth and polished metal, so they present little drag on whatever surface the robot is rolling over. Even so, the ArdBot is best suited for travel on hard surfaces or carpet with a short nap.

The two drive motors run off their own battery supply which is a set of four AA rechargeable or non-rechargeable cells. The motors are standard size radio control airplane servos that have been modified for continuous rotation.

The ArdBot reference design uses servos that come from the factory already modified so you don't have to hack them. I used a pair of GWS S-35 servos, but there are others available (see **Sources**) for under $15 each. I won't provide instructions here on how to modify a servo for continuous rotation. That subject has been tackled in past issues of *SERVO* and *Nuts & Volts*, so I'll leave it at that.

Making the ArdBot Base

The ArdBot is constructed with four body pieces held together with hardware fasteners. **Table 1** provides a full list of mechanical parts. **Tables 2** through **5** specify the other components to complete the ArdBot.

All body pieces assume 1/4" thick material. For your reference, **Figure 1** shows a completed ArdBot, ready to be programmed and played with. The body pieces include:

- *Bottom deck* measuring 7" diameter with cutouts for the wheels (see **Figure 2**). The deck includes a number of holes, of which only six are required. Any other holes are up to you. I've included several additional holes at the front and back of the deck for mounting bumper switches and other sensors. The wheel cutouts measure 2-5/8" x 7-5/8"; sized for commonly available 2-1/2" or 2-5/8" diameter robotic wheels for R/C servo motors.
- *Top deck* measuring 7" x 5" (see **Figure 3**). Only four of its holes are critical; these mate with matching holes in the bottom deck using a set of four standoffs. A 1/2" diameter hole in the center (or thereabouts) provides a throughway for wires from the bottom deck. The other holes as shown are optional, and are for attaching sensors and other accessories.
- *Pair of servo mounts* (see **Figure 4**) for attaching the servos to the bottom deck. You can make these

Table 1. Mechanical Parts.

Qty	Description
1	7" diameter bottom deck with wheel well cutouts for the drive wheels.
1	7" × 5" top deck.
2	Servo mounts.
4	90° plastic L brackets for attaching the servo mounts to the bottom deck. These brackets measure 3/4" × 3/4" with hole centers at 3/8", and are made to work with the two servo mounts.
16	4-40 x 1/2" machine screws and nuts for attaching the servos and servo mounts to the bottom deck.
4	Deck risers consisting of: (4) 1-3/4" aluminum (or plastic) risers with 4-40 threads; (4) 4-40 × 1/2" pan head machine screws; and (4) 4-40 × 1/2" flat head machine screws.
2	Skids consisting of: (2) 8-32 × 3/4" machine screws; (2) 8-32 hex nuts; and (2) 8-32 acorn (cap) nuts.
3	Sets of mounting hardware for Arduino Uno, consisting of (3) 4-40 × 1/2" machine screws; (3) 4-40 nuts; and (3) plastic washers.

* For your convenience, all mechanical pieces — including precut decks and servo mounts — at are available through Budget Robotics. See the **Sources** box for details.

Table 2. Motors and Wheels.

Qty	Description
2	Standard size R/C servo motors, modified for continuous rotation.
2	2-1/2" or 2-5/8 diameter wheels with hubs to attach to the servo motors.

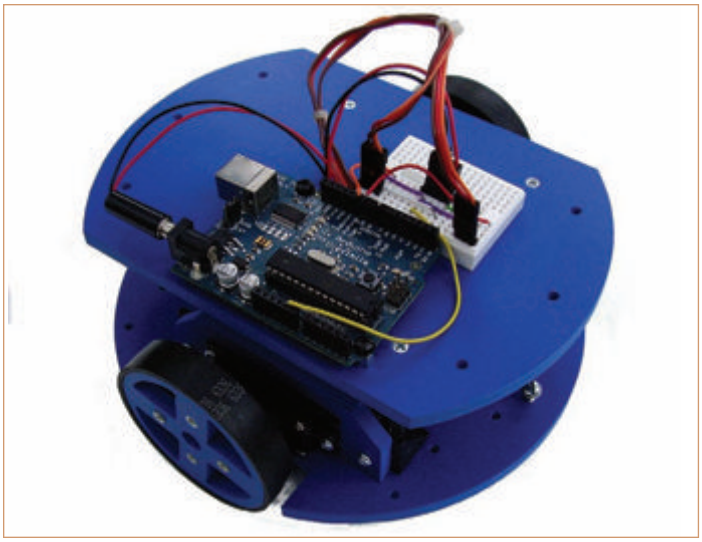


FIGURE 1. The completed ArdBot with Arduino microcontroller board, solderless breadboard, servos, wheels, and all body parts.

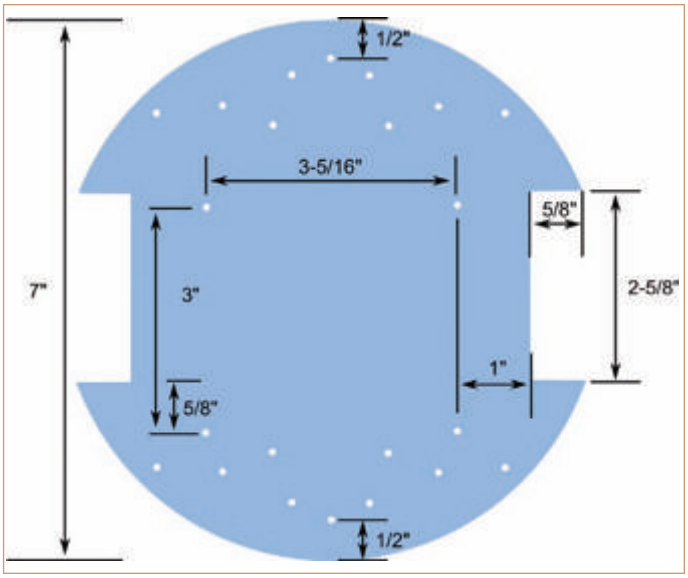


FIGURE 2. Layout pattern for cutting and drilling the bottom deck of the ArdBot. The only truly critical dimensions are the cutouts for the wheels and the placement of the two sets of holes immediately beside the wheel cutouts. These holes are for the servo mounts. See **Figure 5** for a description of all holes.

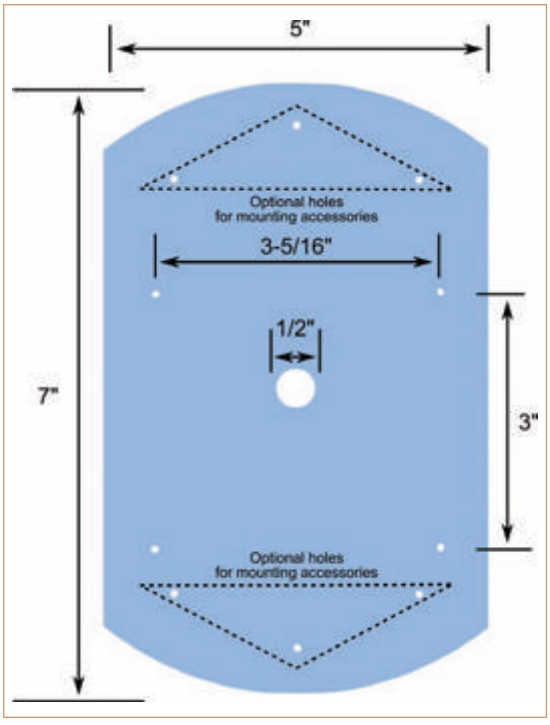


FIGURE 3. Layout pattern for cutting and drilling the top deck of the ArdBot. Critical holes are the four small ones nearest the center. These must match the four servo mounting holes in the bottom deck.

yourself or, if you choose, purchase them separately. If you make the mounts, be aware that sizing is critical. The two holes on either side of the mount must be spaced 3" apart to accommodate the same hole spacing in the bottom deck.

The base parts may be cut from stock measuring 12" x 12" which is a common size for expanded PVC or other plastic purchased through mail order. A motorized scroll saw is the perfect tool for cutting out the ArdBot base components, but if you don't have one handy, a coping saw also works. Use a wood blade; it'll work whether you're making the base with aircraft-grade plywood (available at the hobby store), PVC, or other plastic.

If using foam board or picture mat, you can cut the pieces using a sharp hobby knife or mat cutter. The usual safety precautions apply. A circle cutting jig makes perfect circles when using these materials. If you don't own a circular jig yourself, see if the local picture frame store will make the cuts for you. When using picture mat material, cut two of everything, and double-up the pieces for extra stiffness. Except for the large center hole in the top deck, all holes are drilled with a 9/64" bit.

Assembling the ArdBot

With the body pieces constructed (or purchased) and all other parts in hand, you're ready to build your ArdBot. Here's how.

Step 0

Before assembly, you may want to use 150 grit sandpaper to smooth the edges of the base parts. Orient the bottom deck so that the holes are aligned as shown in **Figure 5**. Note that the holes for each servo are not symmetrically placed on the deck. This is to accommodate

Table 3. Electronic Parts.

Qty	Description
1	Arduino Uno (or compatible) microcontroller board with USB programming cable.
1	Mini solderless breadboard; 170 tie points.
1	Set of solderless breadboard wire jumpers (or make your own using 22 gauge solid conductor wire).
1	AA x four battery holder, with female header connector; see text.
1	Nine volt battery clip, with 2.1 mm polarized barrel plug; see text.
1	Length of 12 (or more) breakaway 0.100" male header pins, double-sided (long) pins; see text.

Table 4. Power.

Qty	Description
4	AA alkaline or nickel metal hybride rechargable batteries.
1	Nine volt battery.

Table 5. Optional (but nice to have) Parts.

Qty	Description
1	Nine volt metal or plastic battery holder.
1	Hook-and-loop (Velcro) strips for mounting battery holders and solderless breadboard; small pieces of double-sided foam tape.

the offset of the servo drive shaft. While there is technically no "front" or "rear" of the ArdBot, for the purposes of assembly, the top of the illustration in **Figure 5** is the front and the bottom is the rear.

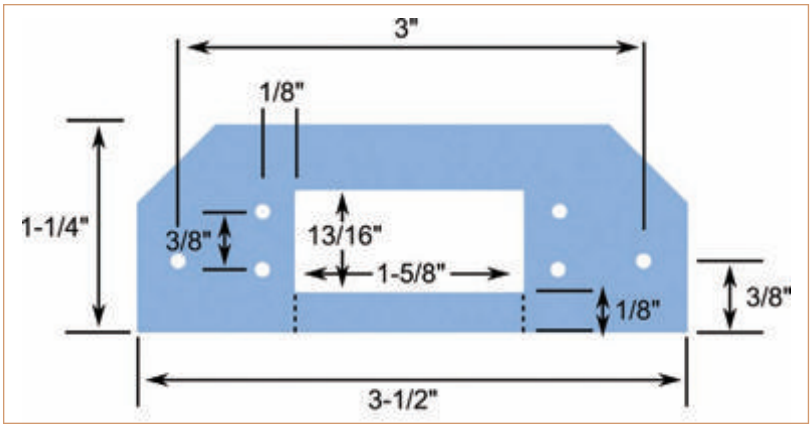


FIGURE 4. Layout pattern for cutting and drilling the servo mount. You'll need two of these. If cutting the inside rectangle proves difficult, you can instead make the mounts by cutting through at the dotted line. The mount will be a little more fragile, so handle it carefully. Use all four screws to secure the servo in the mount, rather than just two.

Step 1

Insert a servo into a servo mount by sliding it back-end first through the mount. The fit may be tight, depending on the make and model of the servo. (As necessary, enlarge the rectangle for the servo using a file or coarse sandpaper.) Do not force the servo into the mount or the mount may be damaged.

Secure the servo to the mount with 4-40 x 1/2" screws and hex nuts (**Figure 6**). You can use four screws for each servo, or only two. When using two screws position them on opposite corners of the servo mounting flange, as shown.

Repeat for the opposite servo and mount. *Be sure to construct the second servo and mount in a mirror image to the first!* Refer to **Figure 9** in Step 3 to see how the motors should be inserted into the mounts. For reference, also see **Figure 12** for an aerial view of the ArdBot and its completed bottom deck.

Step 2

Using 4-40 x 1/2" machine screws and nuts, attach two plastic L brackets to each of the servo mounts (**Figure 7**). You'll be make a "left" and a "right" mount assembly.

For the left mount assembly, the motor shaft should face to the left and toward the "top" of the deck (as referenced in **Figure 5**). Attach the L brackets to the right side of the mount. For the right mount assembly, the motor shaft should face to the right, also toward the top of the deck. Attach the L brackets to the left side of the mount.

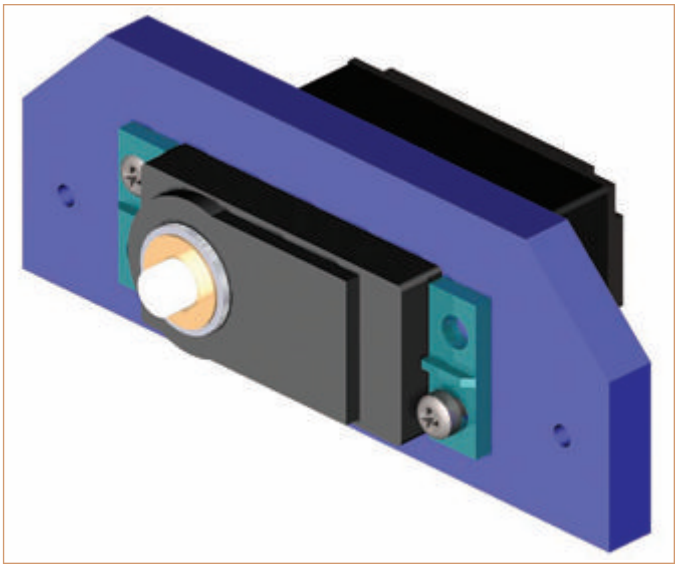


FIGURE 6. Servo motor secured into one of the servo mounts. You need two of these.

Insert the machine screws through the L bracket, then through the servo mount. Secure on the other end with a nut. Before tightening, be sure the bottom of the L bracket is flush with the bottom edge of the servo mount.

Step 3

Attach the left mount assembly to the bottom deck using two 4-40 x 1/2" screws and standoffs. The screws should come up from the underside of the deck, through

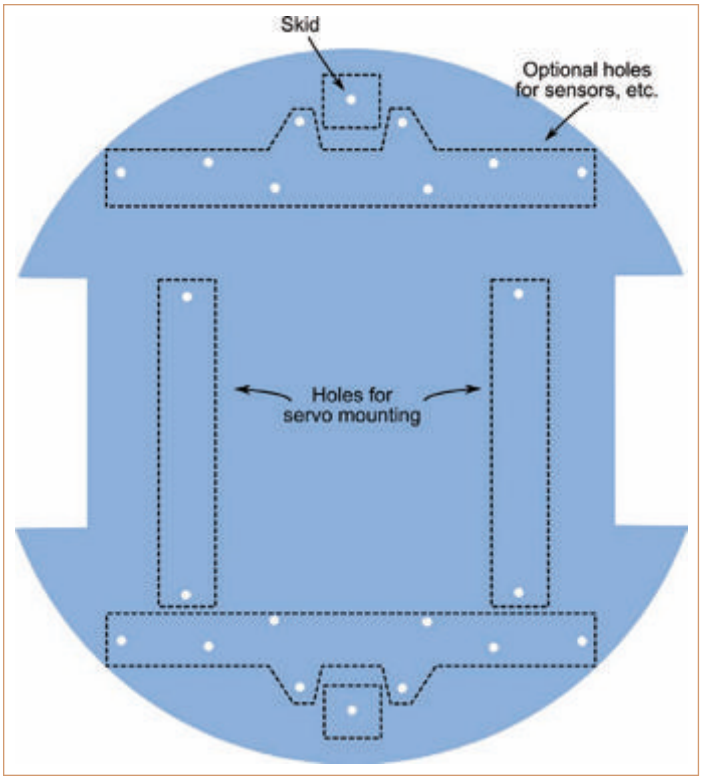


FIGURE 5. Only four holes are critical for the bottom deck: the two sets marked Holes for servo mounting, and the front and rear Skid. The rest are optional for sensors and other accessories you may want to add later.

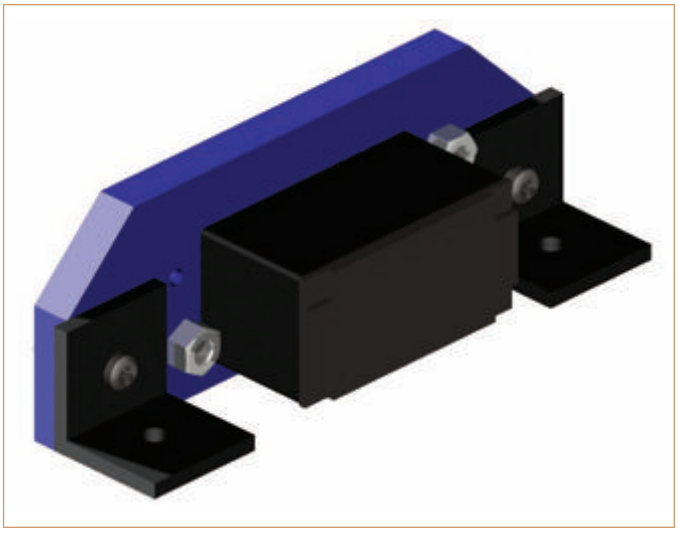


FIGURE 7. Attach two L brackets to the servo mount. The L brackets should be flush with the bottom of the servo mount.

the L bracket, and then into the standoff as shown in **Figure 8**. When orienting the mount assembly, be sure that the servo shaft is centered in the wheel well cutout. Align the assembly so they are parallel with the wheel well cutout, then tighten all the screws. **Figure 9** shows how the completed servo, mount, and standoffs should look. Repeat the same procedure for the right mount assembly.

Step 4

Attach the front and rear skids as shown in **Figure 10**. Each skid uses an 8-32" machine screw, hex nut, and acorn (cap) nut.

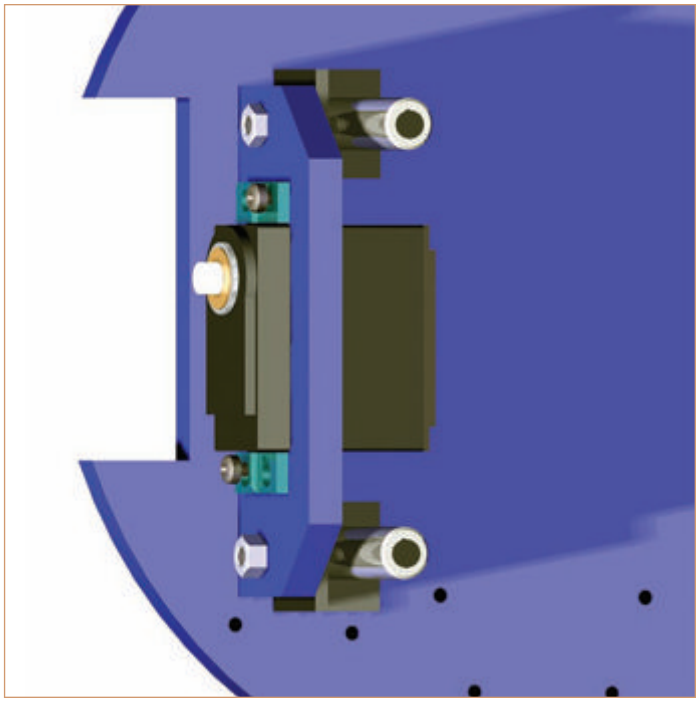


FIGURE 9. Here's how the completed servo mount should look with standoffs in place.

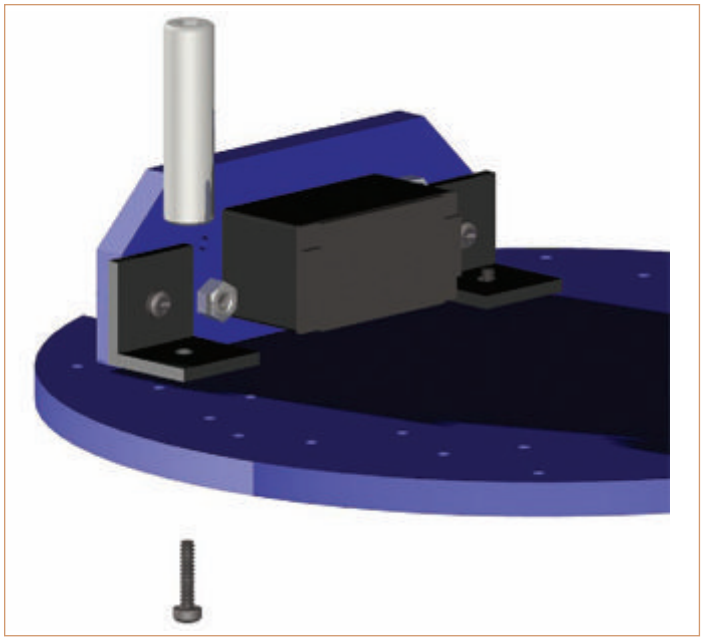


FIGURE 8. Secure the servo mounts to the bottom deck using machine screws and threaded standoffs. The standoffs serve to separate the decks.

1. Using a screwdriver, thread a machine screw into the hole at the front and back of the deck (refer to **Figure 5** for the location of these holes). The screw is inserted from the top of the deck (the side with the servos). The holes for the skids are undersized for 8-32 machine screws. When using a soft material like wood or PVC plastic, the fastener will tap the hole as you screw it in. Continue threading the screw into the hole until the head is about 1/4" from the deck, as indicated in the picture.

2. Put the hex nut onto the screw, followed by the acorn nut. Tighten the acorn nut against the hex nut.

Repeat these steps for the other skid. You may adjust the height of the skid by loosening or tightening the machine screw in the hole. If you need greater height adjustment or the hole for the skid is too large to self-tap,

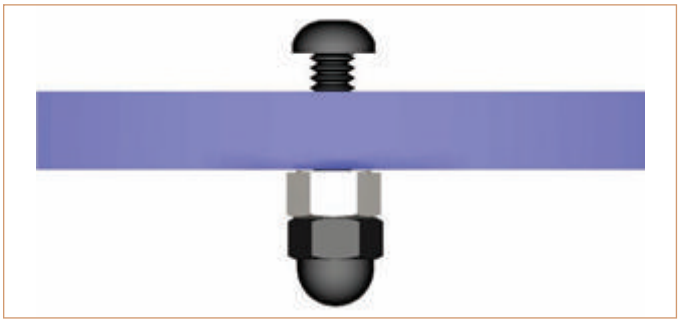


FIGURE 10. ArdBot uses static skids (made with 8-32 metal fasteners) for front and back balance. You can adjust the height of each skid to compensate for the diameter of wheels you use.

FIGURE 11. If you need additional height control for the skids or the hole for the skid cannot be threaded, use a longer 8-32 screw with hex nuts above and below the deck.

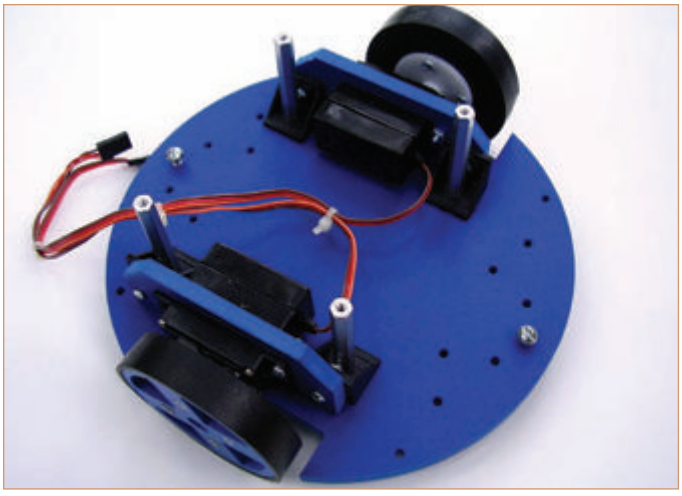


FIGURE 12. The completed bottom deck of the ArdBot. Note the orientation of the servos in the mounts.

merely use a longer machine screw and tighten into place using nuts on both the top and bottom of the deck, as shown in **Figure 11**.

Step 5

Attach the wheels to the servos. Each wheel is secured with a small self-tapping screw that is supplied with the servo. Note that the servo shaft is splined; this spline matches the wheel hub. Be sure to press the wheel onto the shaft firmly while tightening the screw. Do not over-tighten the wheel mounting screw, but be sure the wheel is on snugly. **Figure 12** shows the completed bottom deck of the ArdBot, with motors, mounts, and wheels attached. (I've bound the wire leads for the servos using cable ties to keep things neat. You can do the same if you wish.)

Step 6

Secure the side of the nine volt battery holder against the side of the AA battery holder using a small piece of double-sided foam tape or hook-and-loop (Velcro). Next, secure the AA battery holder to the approximate center of the bottom deck using a square or two of hook-and-loop to keep it in place. Note the electrical connections for both the nine volt battery and the AA battery holder:

- The nine volt battery uses the traditional two-prong battery clip, terminated on the other end with a 2.1 mm barrel plug. This plug inserts into the power jack of the Arduino. You can make this power lead yourself by soldering a barrel plug onto a standard two-prong battery clip, or purchase one ready-made (see the **Sources** box). When constructing your own, be absolutely sure the + (positive) connection is the center of the plug; the – (negative) connection is the outside of the barrel.
- The AA battery holder uses a female 0.100" pin header connector. You can use a connector with two or more pins; the additional pins can be used to help assure proper polarity. With just two pins, you must be VERY careful to never (and I mean NEVER, EVER!) reverse the polarity of the connector. If you do, your servos will be instantaneously and permanently damaged. By using (for example) a four pin connector, you can block up one of the unused terminals. This helps prevent you from reversing the connector when you plug it in. (Of course, still be careful, no matter what system you use!) Insert fresh batteries into the holders and attach the clip to the nine volt battery. The holders with batteries are shown in **Figure 13**.

Step 7

Find a favored spot on the top deck for your Arduino, and mark three holes for mounting the board. Be sure not

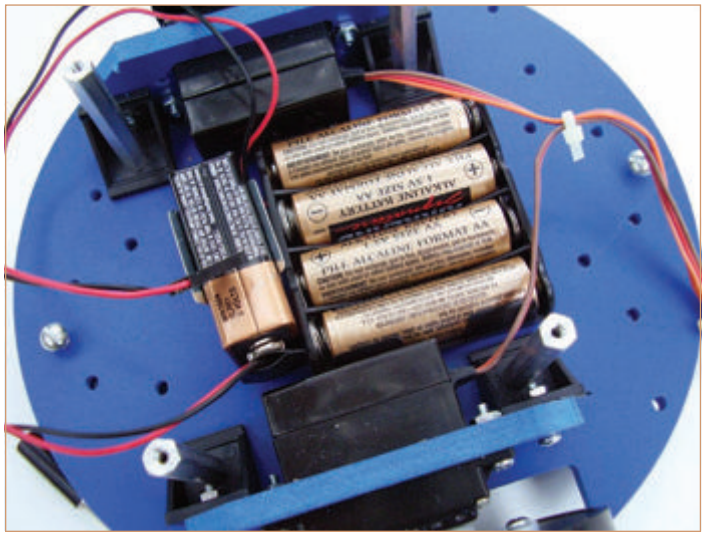


FIGURE 13. The bottom deck is large enough for several battery packs, and they can be neatly placed in the center. The reference design uses a nine volt battery to power the Arduino, and a holder with four AA cells to power the servo motors.

Listing 1

```
/*
 ArdBot ServoTest
 Tests servos of robot by moving them in
 different directions
 Requires Arduino IDE version 0017 or later
   (0019 or later preferred)
*/

#include <Servo.h>

Servo servoLeft;              // Define left servo
Servo servoRight;             // Define right
servo

void setup()
{
   servoLeft.attach(10);      // Set left servo to
                              // digital pin 10
   servoRight.attach(9);      // Set right servo
to
                              // digital pin 9
}

void loop()                   // Loop through
                              // motion tests
{
   forward();                 // Example: move
                              // forward
   delay(2000);               // Wait 2000
                              // milliseconds
                              // (2 seconds)
   reverse();
   delay(2000);
   turnRight();
   delay(2000);
   turnLeft();
   delay(2000);
   stopRobot();
   delay(2000);
}

// Motion routines for forward, reverse, turns,
// and stop
void forward()
{
   servoLeft.write(0);
   servoRight.write(180);
}

void reverse()
{
   servoLeft.write(180);
   servoRight.write(0);
}

void turnRight()
{
   servoLeft.write(180);
   servoRight.write(180);
}

void turnLeft()
{
   servoLeft.write(0);
   servoRight.write(0);
}

void stopRobot()
{
   servoLeft.write(90);
   servoRight.write(90);
}
```

to cover up any of the four holes used for securing the top deck in place. Otherwise, you'll have to remove the Arduino in order to take off the top deck

Drill the three holes using a 9/64" bit. Secure the Arduino board to the top deck using 4-40 machine screws, nuts, and plastic washers. The washers go between the heads of the screws and the board, and minimize the possibility of a short circuit.

Mount the mini solderless breadboard so that it's close to the Arduino, but doesn't block the 1/2" wiring access hole in the top deck. Though most mini breadboards come with double-sided self-adhesive tape, I recommend that you don't use the tape. Instead, mount the board using a square or two of hook-and-loop. This allows you to easily remove the board when you need to.

Step 8

To complete the ArdBot, secure the top deck to the standoffs using 4-40 x 1/2" flat head screws. Assuming you are using a soft material (wood, PVC plastic, foam board, etc.), the heads of the screws should countersink by themselves as you tighten them and lay flush against the deck. Thread the battery and servo leads through the center hole of the top deck. To keep down cost and complexity, there are no power switches for the batteries, so leave the battery leads unattached until you're ready to program and use the ArdBot. (When you're done playing, be sure to unplug the batteries to keep them from draining.)

Two-Servo Wiring Plan

The Arduino lacks direct connections for attaching the servo motors. Instead, the mini breadboard provides prototyping space for connecting up both servos, as well as the AA battery holder that powers the servos. Refer to **Figure 14** (schematic) and **Figure 15** (pictorial) for wiring the solderless breadboard. Using a strip of 0.100" double-sided (long) male header pins, break off two sets of three pins, and one set of pins for the AA battery connection.

Note that you want the version of male header pins that are "double-sided" — they're long on both sides. If you use the standard header pins, the length of pins on one side is shorter. These don't make good contact when used with solderless breadboard designs. See the **Sources** box for a couple of mail order companies offering double-sided long header pins. In a pinch, you can use right-angle header pins instead and straighten them out so that all the pins are flat. The reference design uses a AA battery holder with a four-pin female connector. The + and – leads are on the two outside positions of the connector. I've broken off the pin right next to the + connection of the male header, then used a short piece of solid conductor hookup wire to fill in its corresponding hole in the connector. This prevents the connector from being reversed when plugged in.

When wiring the solderless breadboard, be especially careful not to mix positive and negative leads to the servo. Reversing the power leads to a servo will permanently

damage it. *Here's an important note*: The ArdBot uses separate battery supplies for the Arduino and the two servos. In order for everything to function properly, the ground connections for the Arduino and the servo battery supply must be connected together. This is shown in both the schematic and pictorial circuit views.

Make sure to also properly orient the connectors for the servos when you plug them into the board. Servo power leads are color-coded, but the colors aren't universal.

- Ground (–) is typically black or brown.
- Power (+) is most often red, and with modern servos is always in the middle.
- Signal is white, yellow, or sometimes orange (but take care — on some servos the power wire is orange!).

When in doubt, check the spec sheet that comes with your servos. Don't guess!

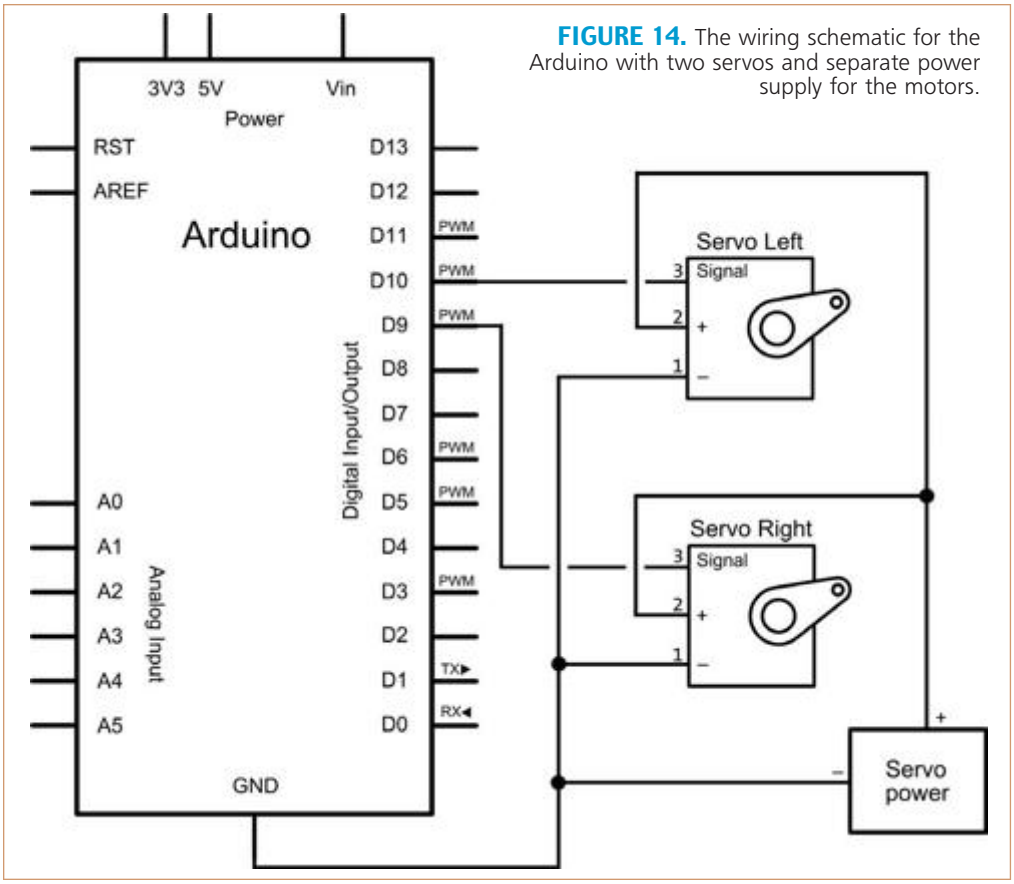


FIGURE 14. The wiring schematic for the Arduino with two servos and separate power supply for the motors.

Servo Test Sketch

With the ArdBot constructed and the breadboard wired, you're ready to test the robot and put it through its paces. Refer to **Listing 1** for a quick servo test sketch.

Start the Arduino IDE, connect a USB cable between your computer and the Arduino (as noted on the Getting Started pages of the Arduino website), and type the program as shown. When done, Verify (compile) the sketch and look for any syntax errors. If there are none, download the sketch to your Arduino.

Once downloaded, put a small book under your ArdBot to lift its wheels off the ground. Disconnect the USB cable, and — in this order — plug the AA battery connector into the breadboard, then plug in the nine volt power to the Arduino power jack. (If you are using an Arduino Diecimila, be sure to switch over the power selection jumper from USB to EXTernal.) If everything is connected properly, the servo motors should go through a test pattern.

Assuming the motors are working as they should, depress the Reset switch on the Arduino board and place the ArdBot on the ground. Release the switch and the robot should perform its self-test maneuvers. If the motors aren't moving, double-check your wiring, making sure the servo connectors are properly oriented. They won't work if the connectors are reversed on the breadboard.

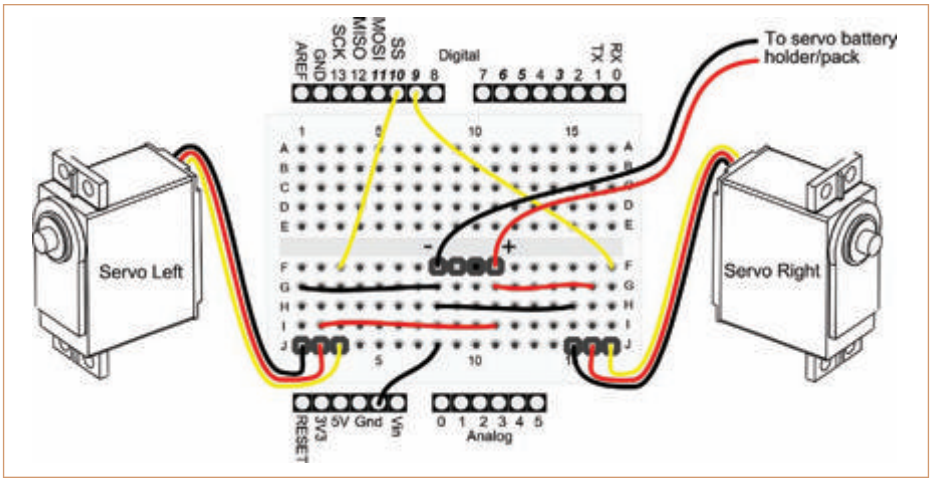


FIGURE 15. Pictorial view of how to connect the Arduino to the two servo motors. Note that the Arduino ground connection is shared with the power for the servos. This is very important.

Sources

Arduino
www.arduino.cc
Prefabricated ArdBot body pieces with all construction hardware.

Budget Robotics
www.budgetrobotics.com

AdaFruit
www.adafruit.com

HVW Tech
www.hvwtech.com

Jameco
www.jameco.com

Parallax
www.parallax.com

Pololu
www.pololu.com

Robotshop
www.robotshop.com

Solarbotics
www.solarbotics.com

SparkFun
www.sparkfun.com

Please note! The list of sources is not exhaustive, and is merely designed to get you started in the right direction. There are other companies who sell these items, and not all sources are listed. Common parts like battery holders and breadboard jumper wires are not included here, as they are readily available at RadioShack and hundreds of online electronics supply stores.

Check out ***www.fritzing.com*** for a user-to-user Arduino project community, including an Arduino development library that allows you to create virtual breadboard designs of your projects. You may then turn your projects into schematics and even etchable circuit boards. We've used Fritzing to prepare some of the illustrations for this series of articles.

Closer Look at the Test Sketch

Before closing out this month's installment of the ArdBot, let's quickly review how the test sketch works. First off is an *include* statement to the *Servo.h* library header file which is provided with the Arduino IDE installation. This file and its corresponding C language program, provide all the actual coding to make the servos function.

Next comes two statements that create, or *instantiate*, two *Servo* objects for use in the remainder of the sketch. Each object represents a physical servo attached to the Arduino. Methods of these objects include things like specifying which digital pin is used to connect from the Arduino to the servo, and the position of the servo. Note I've given the two *Servo* objects descriptive names: *servoLeft* and *servoRight*. It's easier to keep track of things this way.

In the *setup* function, the *servoLeft* and *servoRight* objects are "wired" to their respective pins on the Arduino; in this case, pin 10 for *servoLeft* and pin 9 for *servoRight*.

Now comes the main body of the program, provided in the *loop* function. It contains a series of user-defined functions for *forward*, *backward*, and so on, plus a delay of 2,000 milliseconds (two seconds) between each function. You can see that the robot repeats the same demonstration steps over and over:

- Goes forward for two seconds.
- Reverses for two seconds.
- Turns right for two seconds.
- Turns left for two seconds.
- Stops for two seconds.

Finally, each user-defined function specifies the specific motion to apply to the servos. With the *Servo* object, servos are commanded to move one direction or another by (among other ways) specifying an angle between 0 and 180. The servo then moves to that angle in response.

When using servos that have been modified for continuous rotation, 0 makes the servo rotate one direction; 180 makes the servo rotate in the opposite direction; and 90 makes it stop. Pretty easy, isn't it?!

In our next installment, we'll look at servo programming in depth, as well as connecting some sensors to the ArdBot for reactive control, getting feedback from the robot, and more! **SV**

Gordon McComb can be reached at rduino@robotoid.com.

Main Components Sources

This is a *selected* list of North American sources for the main components for the ArdBot.

Arduino Duo or Duemilanove

Source	Item or SKU
Adafruit	50
HVW Tech	28920 (Freeduino SB)
Jameco	2121105
RobotShop	RB-Ard-03
Pololu	1616
SparkFun	DEV-09950

Solderless Breadboard; 170 tie-points

Source	Item or SKU
Adafruit	65
HVW Tech	21380
Jameco	2109801
Parallax	700-00012
RobotShop	RB-Spa-139

Nine volt to 2.1 mm Barrel Plug Cable

Source	Item or SKU
Adafruit	80
SparkFun	PRT-09518

Continuous Rotation Servo (Futaba spline)

Source	Item or SKU
Parallax	900-00008
Pololu	1248
RobotShop	RB-Gws-23
Solarbotics	36000
SparkFun	ROB-09347

2-1/2" or 2-5/8" Rubber Wheels (Futaba spline)

Source	Item or SKU
Adafruit	167
HVW Tech/ Solarbotics	SW
Parallax	28109
Pololu	226
RobotShop	RB-Sbo-86

Double-sided (long) Male Header Pins

Source	Item or SKU
Parallax	451-00303
Pololu	1065

By Daniel Ramirez

VEX Stepper Motor Control Experiments - Part 2

Building the VEX SunBot

Ancient Babylonians, Egyptians, Aztecs, Mayans, Greeks, and Stonehenge Druids have been following the sun, moon, and star positions for centuries. These vital details gave them seasonal information about when they could plant and harvest their crops and perform religious ceremonies. It was during the height of the Renaissance that the planetary orbits and the sun and moon positions could be computed using the great discoveries in orbital mechanics and gravitation discovered by Galileo Galilei, Nicolaus Copernicus, Tycho Brahe, Johannes Kepler, Sir Isaac Newton, and many other astronomers of that era. These orbital mechanical equations have been refined over the latest centuries to include the effects of Einstein's Theory of Relativity and Einstein's general theory of relativity so that now mankind can predict the planetary orbits and the sun's position with incredible accuracy.

SunBot II's mission is to collect critical information needed from our nearest star — the sun! This insolation data is required to plan for and design the next generation of solar panels and solar cells that will provide environmentally clean power alternatives to fossil fuels now and in the foreseeable future. Reducing or eliminating carbon dioxide (CO_2) emissions is important and solar technologies can play a big part in the solution to this.

SunBot II will be a self-powered, VEX-based robot whose mission is to track the sun anywhere on earth under all lighting conditions (during sunny or dark and cloudy days), using astronomical tables to collect critical solar data necessary to produce solar power generation systems by acting as the guide for hundreds of solar panels located within a one mile radius of it. SunBot also demonstrates how VEX starter kits can be used to develop prototype environmentally-friendly products.

Although there are many kinds of solar panels currently sold worldwide by various vendors, the most common configuration is the fixed position panel which is typically positioned facing south and oriented at a 35 to 45 degree angle facing the sun. This position allows solar cells to collect sunlight during most of the day, but misses out on the sun's direct rays at noon. You see these kinds of panels in solar powered (PV) homes, businesses, and even along highways. Existing motorized panels are more expensive due to the motors and controllers required, but they can track the sun all day long. These types of panels are usually found in solar energy farms, universities, and energy research facilities worldwide where sunlight is plentiful all year long. Commercially sold solar panels usually use two or more photocells to find the brightest spot in the sky and track it. Bad weather, cloudy, dark, and rainy days can cause some of the tracking mechanisms to hunt for the sun excessively, wasting previously stored battery energy by having the motors move unnecessarily. So, what does this project have to do with stepper motors and VEX? With the Gulf Coast oil disaster now behind us, I believe it's time to start thinking more about alternative forms of energy.

The prototype SunBot was assembled using VEX components (including standard VEX motors) and is mounted on my VEX-based Gilbert IV Explorer robot shown in **Figure 1**. The problem with this prototype was that the gearing used did not provide enough pointing accuracy which is why I decided to switch from VEX motors to stepper motors for the azimuth and elevation drive.

VEX SunBot Features

The Gilbert IV robot shown in **Figure 1** is the mobile

FIGURE 1. This prototype SunBot was assembled using VEX components and is mounted on my VEX based Gilbert IV Explorer robot shown in this photo. The VEX motors are being replaced with very accurate stepper motors to improve its solar tracking abilities.

platform for SunBot which also houses the delicate electronics. It can move it around the yard using its 6WD drive motion subsystem while at the same time be able to point the marine solar panel in two degrees of freedom (azimuth and elevation). I plan to replace the current SunBot hardware that includes the solar panel, mast, and azimuth drive assembly with SunBot II shown in **Figure 2**. Don't worry – you won't have to build the robot platform for SunBot II since it can be separated from the mobile robot and mounted on a fixed platform. The complete SunBot II system shown in **Figure 2** includes the SunBot II robot frame and controller assembled from VEX components, surplus stepper motors, SparkFun EasyDriver controllers, and a 12 volt solar panel, VEX limit switches, and VEX Quadrature Optical Encoders. With this hardware, it should be possible to:

- Point the solar panel 0-360 degrees azimuth, within ± 1 degree resolution.
- Tilt the solar panel ± 90 degrees elevation, within ± 1 degree resolution.
- Use GPS to obtain SunBot's latitude, longitude, and altitude.
- Use astronomical or navigational tables to obtain the sun's azimuth and elevation for a particular time interval, and command the stepper motors to go to that position.

In Part 1, we provided the introduction to using stepper motors with a ULN2803 IC. I also mentioned that the ULN2803 could be used to drive other electromechanical components including relays and solenoids. This particular application of the ULN2803 will be covered in a future article. In Part I, we also mentioned that the SparkFun EasyDriver board was going to be used to upgrade the HERO 1 robot electronics which coincidentally is also being used on SunBot II. Here we use the more efficient EasyDriver stepper motor driver since the original ULN2803 IC is not powerful enough to drive the larger geared stepper motors used for the SunBot II azimuth and elevation drives. We will also show how you can micro-step these motors to obtain even more positional accuracy, using the advanced micro-stepping feature that is available on the EasyDriver board. These boards are reasonably priced at around $15 each which is a great deal considering the functionality they bring to VEX.

We will demonstrate using the VEX microcontroller with the EasyDriver boards and surplus stepper motors, and a 12 volt solar panel in order to rapid prototype SunBot II. You should be able to carry out many interesting astronomical and solar energy experiments at home or in school with the information presented in this article. Even if you don't want to build the complete system, you can still use the information to drive stepper motors for any of your own projects that need precise stepper motor control using micro-stepping techniques.

Power collected from the solar panel is used to charge the 12 volt SLA in order to keep SunBot II in motion all day long. One problem I have is regulating the power collected so that it does not overcharge

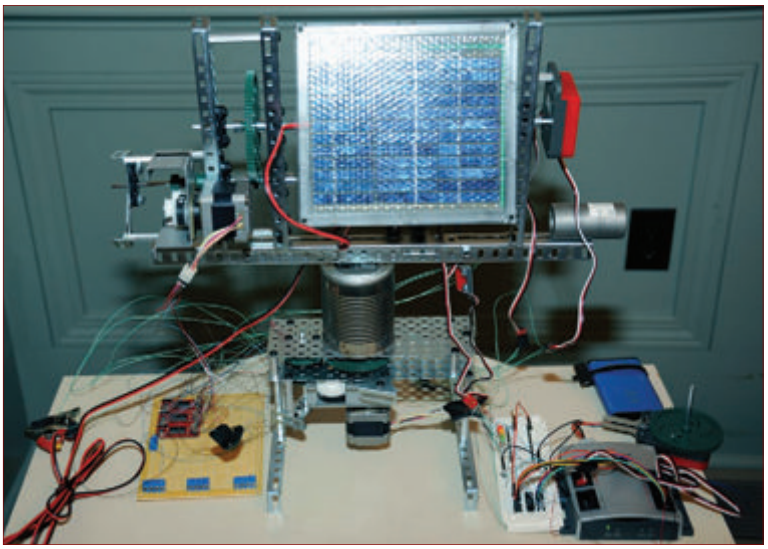


FIGURE 2. The SunBot II robot frame and controller is assembled from VEX components including metal parts, VEX limit switches, VEX quadrature optical encoders, surplus stepper motors, and SparkFun EasyDriver controllers, and a 12 volt solar panel.

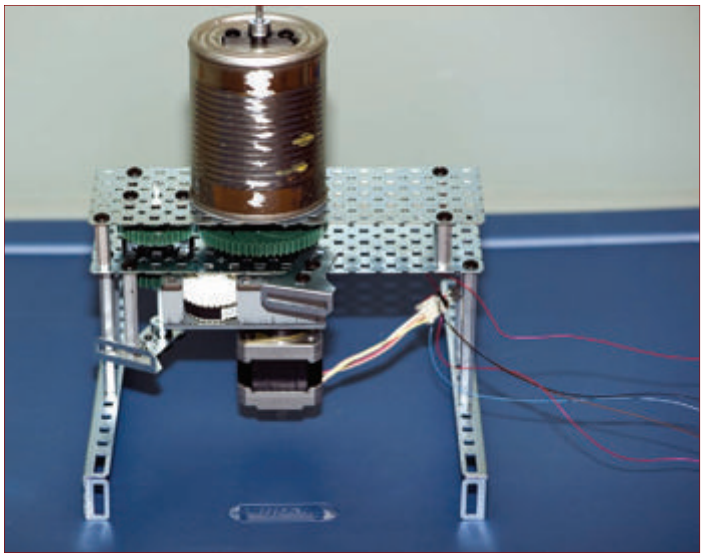


FIGURE 3. The SunBot II azimuth assembly is made from VEX components, a recycled tin can, and the azimuth geared stepper motor assembly.

the battery and damage it. There are solar power voltage regulators commercially available for larger installations. One nice feature of this implementation is that it will not depend on using photocells or phototransistors to find the brightest spot in the sky in order to keep the panel pointed at the sun. Instead, I plan to use astronomical data found in navigation tables of azimuth and elevation positions for a given time interval, as mentioned previously.

The complete SunBot II azimuth assembly shown in **Figure 3** is made from VEX components, a sturdy recycled tin can, and the azimuth geared stepper motor assembly. The elevation frame support column and azimuth drive are also shown. Since cylinders and pipes are not currently part of the VEX structure inventory, I adapted a tin can to support the solar panel assembly so that it could rotate 0 to 360 degrees. A VEX limit switch mounted on the elevation axis is used to stop the rotation so that the wires don't get all wound up. The azimuth drive uses another geared stepper motor mounted on a tin can. The tin can was adapted by simply drilling two holes on the top to allow a VEX bracket to be mounted using standard VEX nuts and bolts. (See **Figure 4**.) Other household items such as steel or PVC pipes, metal boxes, etc., can be adapted for use with VEX components, as well.

Recall another VEX limit switch is mounted on the azimuth axis to stop the rotation so that the wires don't get all wound up. The firmware running on the VEX microcontroller is used to poll the limit switches and stop the stepper motors if the travel limits are exceeded.

Sunbot II's elevation drive shown in **Figure 5** is used to tilt the solar panel 0 to 90 degrees using a stepper motor adapted to a VEX gear box. The VEX controller will drive the stepper motor to position the panel. Notice how the stepper motor gears meshed perfectly with the VEX gears. Not all adaptations are this easy but sometimes you find that hardware from various sources has the same form factor. These same geared stepper motors can be used for other robot applications including odometry, dead reckoning, TurtleBots, and even 3D plotters that require very precise motions.

SparkFun EasyDriver Stepper Motor Driver

The EasyDriver shown in **Figure 6** is a simple to use stepper motor driver, compatible with anything that can output a digital 0V to 5V pulse. EasyDriver requires a 7V to 30V supply to power the motor, and has an onboard voltage regulator for the digital interface. Connect a four-wire stepper motor and a microcontroller, and you've got precision motor control! EasyDriver drives bi-polar motors, or motors wired as bi-polar, i.e., four, six, or eight wire stepper motors. Note that the microstep select (MS1 and MS2) pins of the A3967 are broken out allowing adjustments to the microstepping resolution. The sleep and enable pins are also broken out for further control.

Remember, I used the EasyDriver board to drive the stepper motors from the VEX microcontroller. Each board requires two of the VEX controller's I/O pins. One pin is used to toggle the step while the other pin is used to change the direction (clockwise or counter-clockwise). A one-step pulse moves the stepper motor one, 1/4, or 1/8 of a step (micro-step), depending on how the EasyDriver is configured using the MS1 and MS2 pins. I used the 1/8 of a step since most low cost surplus stepper motors have a

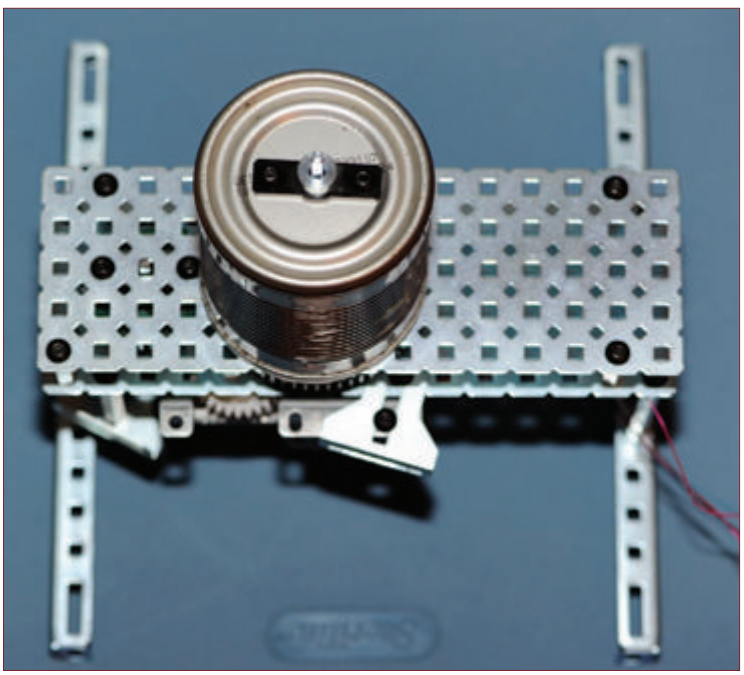


FIGURE 4. Another view showing how we adapted the tin can for the azimuth geared stepper motor assembly. The tin can was adapted by simply drilling two holes on the top to allow a VEX bracket to be mounted using standard VEX nuts and bolts.

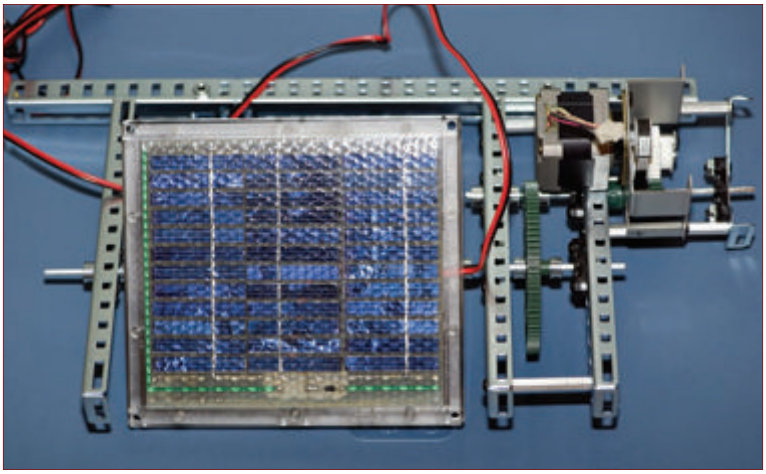


FIGURE 5. The SunBot II elevation assembly is made from VEX components and the elevation geared stepper motor assembly which drives the 12 volt solar panel.

resolution of 1.8 degrees per step, or 200 steps for one complete revolution. Using the 1/8 micro-stepping mode gives me a total of 200/(1/8) = 1,600 steps.

Caution: Do not connect or disconnect a motor while the driver is energized. This will cause permanent damage to the EasyDriver.

To connect the EasyDriver boards and VEX limit switches to the VEX microcontroller for driving the azimuth and elevation stepper motors, use the schematic shown in **Figure 7**. In order to simplify the connections, I used .100 pin headers soldered to the EasyDriver boards as posts for wire wrapping to the microcontroller ANALOG/DIGITAL I/O block. I also used wire wrap to extend the limit switch connections and quadrature optical encoder connections.

CAUTION: When operating SunBot, safety should be your primary concern when working with stepper motors. Also be sure to wear safety eyeglasses, and keep clothing and jewelry away from it when running it. Consider connecting a dedicated pushbutton switch (panic button) that will immediately cut power to the stepper motors in the event of an emergency. Stepper motors can draw a lot of current, so it is wise to use the proper gauge wires.

Construction Details

It is not necessary to use high pointing Swiss precision found in modern "Goto" telescope mounts used by astronomers, or even the heavy duty mounts found in solar energy research centers since this is a low cost alternative. The sun's azimuth and elevation need only to be pointing in the general direction and be accurate to between 0.5 and one degree in either axis and still get good results. The parts that I used to build SunBot II are shown in **Table 1**. Careful assembly and alignment of the metal parts is necessary to insure good results. When VEX parts are not available, we can adapt common parts found in hardware or surplus electronics stores, or even common household items.

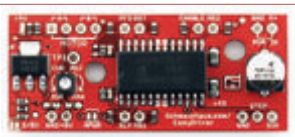


FIGURE 6. The EasyDriver shown is a simple to use stepper motor driver, compatible with anything that can output a digital 0V to 5V pulse.

Results

SunBot in motion was fun to watch as it swept the solar panel in a circle while at the same time tilted the solar panel and demonstrated how stepper motors could be micro-stepped. This version worked reasonably well considering the materials used, but it still needs more refinements. For instance, the azimuth and elevation stepper counts did not consistently correlate to the quadrature encoder counts.

Other problems included exceeding the hard travel limits by not reacting fast enough to the VEX limit switches. These

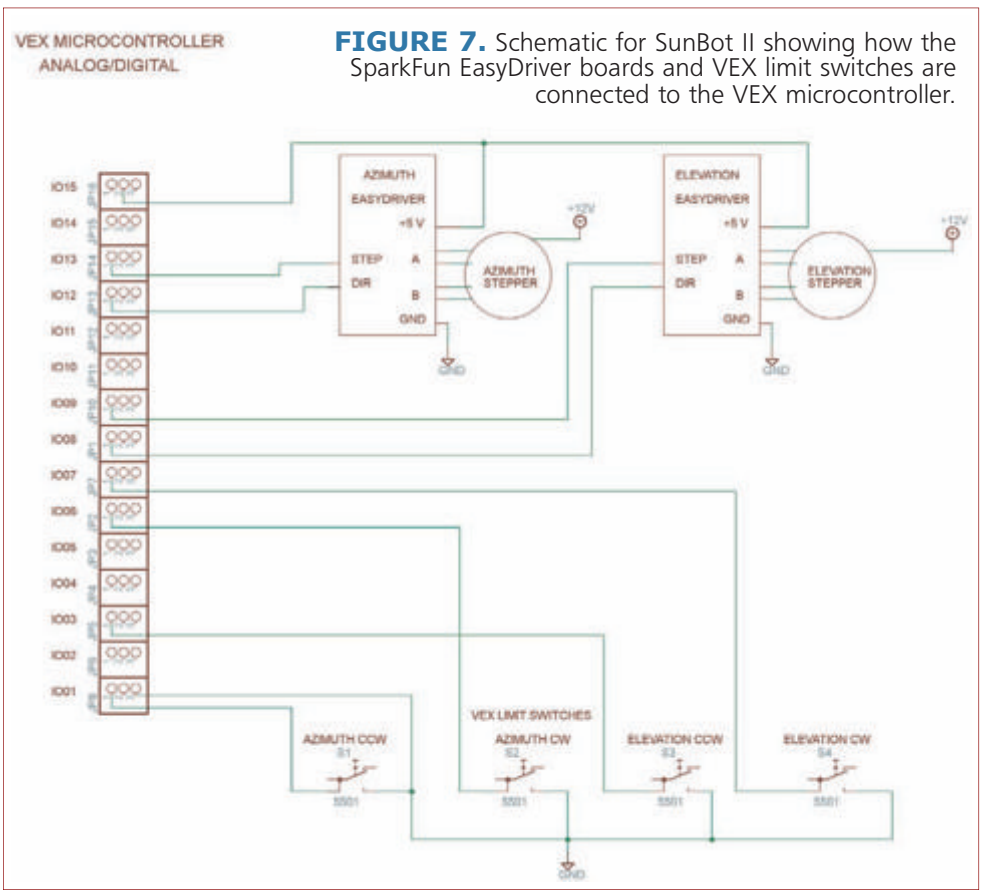


FIGURE 7. Schematic for SunBot II showing how the SparkFun EasyDriver boards and VEX limit switches are connected to the VEX microcontroller.

TABLE 1. *Bill of Materials that I used for assembling the SunBot II project. You may substitute parts as necessary. The VEX quadrature optical encoders are optional and will be used in a future article to calibrate the stepper motor commands to azimuth and elevation positions.*

ITEM	QTY	DESCRIPTION	SOURCE
1	1	VEX microcontroller	Innovation First, Inc. **www.vexforum.com**
2	1	12 volt SLA battery	RadioShack **www.radioshack.com**
3	2	SparkFun EasyDriver stepper controllers	SparkFun **www.sparkfun.com**
4	1	Wire-wrap cable	RadioShack **www.radioshack.com**
5	1	Package of jumper cables	SparkFun **www.sparkfun.com**
6	4	VEX limit switches	Innovation First, Inc. **www.vexforum.com**
7	2	VEX quadrature optical encoders	Innovation First, Inc. **www.vexforum.com**
8	1	VEX structural components	Innovation First, Inc. **www.vexforum.com**
9	1	VEX 9.6 volt battery	Innovation First, Inc. **www.vexforum.com**
10	1	VEX RC transmitter	Innovation First, Inc. **www.vexforum.com**
11	1	VEX receiver	Innovation First, Inc. **www.vexforum.com**
12	2	Geared stepper motors	All Electronics **www.allelectronics.com**
13	1	Heavy duty 12 volt marine solar panel	All Electronics **www.allelectronics.com**
14	1	Recycled tin can (cylinder)	

issues were probably caused by backlash from the gears and also by not detecting the hard stops accurately using the VEX limit switches.

Some of these problems could be corrected by calibrating the movements, or by using a different stepper motor drive mechanism other than gears. Timing belts and direct drive are possible alternatives to fix this problem. In addition, the VEX firmware needs more work to be able to track the sun correctly, once the mechanical problems have been ironed out.

Wrap-Up

Using simple VEX hardware with some PIC18 C firmware, precise and repeatable movements are possible for your next robotics or automation project, using the information in this article. Stepper motors need not remain a mystery to use if you build the SunBot II or similar applications. The SunBot II robot demonstrates very complicated micro-stepping used to aim a solar panel at the sun using stepper motors.

In the next installment, I plan to show a PIC18 C application that I actually used to move the solar panel, and some issues that I encountered with it. I will also cover using the quadrature optical encoders and limit switches, and calibrating the stepper motors which you will need to start tracking the sun with your own SunBot II.

Until next time, happy stepping! **SV**

Give the Gift of SERVO!

Give a one year (12 issues) subscription to a friend and they will receive an extra THREE FREE ISSUES! That's right! A total of 15 issues in all!

U.S. Price:
1 Yr – $24.95

Subscribe online at:
www.servomagazine.com
or call:
877-525-2539 (toll free)
818-487-4545 (outside US)

Not only will they get 15 great issues, but they will also receive:

- **Access to the digital edition**
- **Discounts in the Webstore**
- **Online only content and downloads**
- **And so much more!**

Go now to
www.servomagazine.com
to enter your gift order!

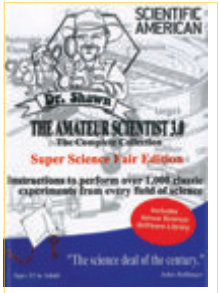
Be sure to use promotion code ***Y0WXMS*** *when ordering.*

The SERVO Webstore

Attention Subscribers ask about your discount on prices marked with an *

CD-ROM SPECIALS

The Amateur Scientist 3.0 The Complete Collection
by Bright Science, LLC



There are 1,000 projects on this **CD**, not to mention the additional technical info and bonus features. It doesn't matter if you're a complete novice looking to do your first science fair project or a super tech-head gadget freak; there are enough projects on the single CD-ROM to keep you and 50 of your friends busy for a lifetime!
Reg $26.95 Sale Price $23.95

ROBOTICS

PIC Robotics
by John Iovine

Here's everything the robotics hobbyist needs to harness the power of the PICMicro MCU!

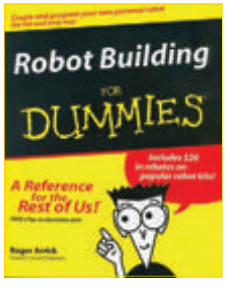
In this heavily-illustrated resource, author John Iovine provides plans and complete parts lists for 11 easy-to-build robots each with a PICMicro "brain." The expertly written coverage of the PIC Basic Computer makes programming a snap – and lots of fun.
$24.95

FIRST Robots: Rack 'N' Roll: Behind the Design
by Vince Wilczynski, Stephanie Slezycki

More than 750 photographs!

The second annual book highlighting the creativity and process behind 30 winning robot designs from the 18th annual international FIRST Robotics Competition. The FIRST organization, founded by Dean Kamen (inventor of the Segway), promotes education in the sciences, technology, and engineering.
Reg $39.95

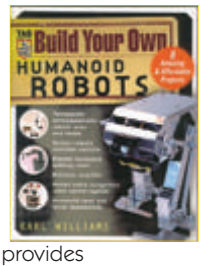
Robot Building for Dummies
by Roger Arrick / Nancy Stevenson



Discover what robots can do and how they work. Find out how to build your own robot and program it to perform tasks. Ready to enter the robot world? This book is your passport! It walks you through building your very own little metal assistant from a kit, dressing it up, giving it a brain, programming it to do things, even making it talk. Along the way, you'll gather some tidbits about robot history, enthusiasts' groups, and more.
$24.95

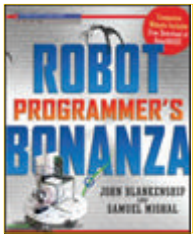
Build Your Own Humanoid Robots
by Karl Williams

GREAT 'DROIDS, INDEED!



This unique guide to sophisticated robotics projects brings humanoid robot construction home to the hobbyist. Written by a well-known figure in the robotics community, *Build Your Own Humanoid Robots* provides step-by-step directions for six exciting projects, each costing less than $300. Together, they form the essential ingredients for making your own humanoid robot. **$24.95***

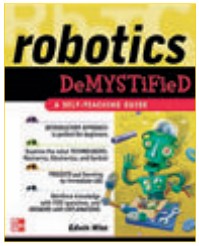
Robot Programmer's Bonanza
by John Blankenship, Samuel Mishal



The first hands-on programming guide for today's robot hobbyist!
Get ready to reach into your programming toolbox and control a robot like never before! *Robot Programmer's Bonanza* is the one-stop guide for everyone from robot novices to advanced hobbyists who are ready to go beyond just building robots and start programming them to perform useful tasks.
$29.95

Robotics Demystified
by Edwin Wise

YOU DON'T NEED ARTIFICIAL INTELLIGENCE TO LEARN ROBOTICS!

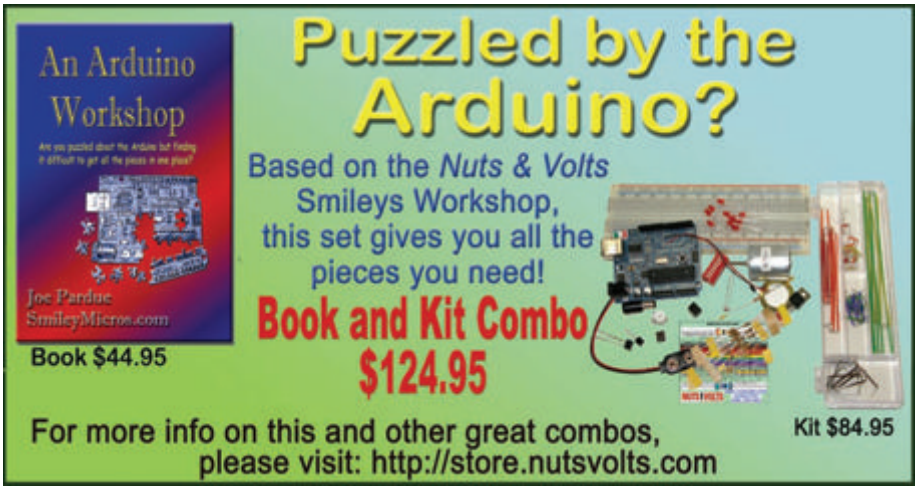


Now anyone with an interest in robotics can gain a deeper understanding — without formal training, unlimited time, or a genius IQ. In *Robotics Demystified*, expert robot builder and author Edwin Wise provides an effective and totally painless way to learn about the technologies used to build robots! **$19.95**

We accept VISA, MC, AMEX, and DISCOVER
Prices do not include shipping and may be subject to change.

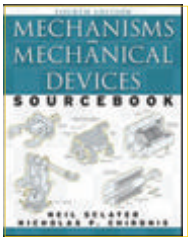
Or order online
www.servomagazine.com

SPECIAL OFFERS



Mechanisms and Mechanical Devices Sourcebook

by Neil Sclater, Nicholas Chironis



Over 2,000 drawings make this sourcebook a gold mine of information for learning and innovating in mechanical design. Overviews of robotics, rapid prototyping, MEMS, and nanotechnology will get you up to speed on these cutting-edge technologies. Easy-to-read tutorial chapters on the basics of mechanisms and motion control will introduce those subjects to you. **Reg $89.95 Sale Price $69.95**

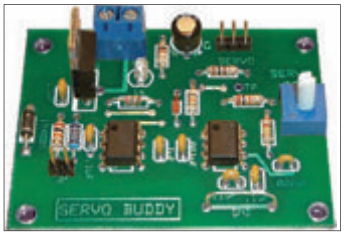
Forbidden LEGO

by Ulrik Pilegaard / Mike Dooley

Forbidden LEGO introduces you to the type of free-style building that LEGO's master builders do for fun in the back room. Using LEGO bricks in combination with common household materials (from rubber bands and glue to plastic spoons and ping-pong balls) along with some very unorthodox building techniques, you'll learn to create working models that LEGO would never endorse. **Reg $24.95 Sale Price $19.95**

PROJECTS

The SERVO Buddy Kit



An inexpensive circuit you can build to control a servo without a microcontroller.

For more information, please check out the May 2008 issue or go to the SERVO websore.

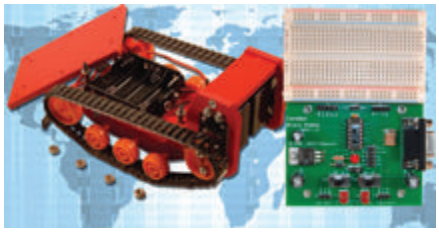
Includes an article reprint.
Subscriber's Price **$39.55**
Non-Subscriber's Price **$43.95**

16-Bit Micro Experimenter Board

Ready to move on from eight-bit to 16-bit microcontrollers? Well, you're in luck! In the December 2009 *Nuts & Volts* issue, you're introduced to the 16-Bit Micro Experimenter. The kit comes with a CD-ROM that contains details on assembly, operation, as well as an assortment of ready-made applications. New applications will be added in upcoming months.

Subscriber's Price **$55.95**
Non-Subscriber's Price **$59.95**

Tankbot Kit & Brain Alpha Kit



Tankbot/Brain Alpha originally by Ron Hackett
Now with New Columnist Calvin Turzillo
A series filled with projects and experiments to challenge you through your learning process while you grow your fully expandable Brain Alpha PCB!
The brain is a PICAXE-14A!
For more info & pictures, visit the *SERVO* Webstore.
Tankbot and the Brain Alpha Kit can be purchased separately.
Combo Price $ 138.95

Propelled By The Propeller Chip

By John Blankenship & Samuel Mishal

The Propeller chip from Parallax, Inc., is an innovative invention and a significant advancement in the microcontroller arena. It incorporates eight parallel processors (called cogs), and with its very easy to learn and powerful language called Spin, you can implement amazing projects that would be beyond the abilities of most controllers in its price range.

A while back, we implemented a project to experiment with PID attitude control of a space station model (**Figure 1**). The overall system was quite complex and constituted many intricate subsystems. The quadrature encoder aspect of the project was the subject of an article in the February '09 issue of *SERVO Magazine*. You can also see a video of the system in action on YouTube (search for *RobotBASIC Control Application*).

The project required parallel processing and was originally implemented using three individual processors. Recently, we began experimenting with the Propeller chip and decided to redo the project using it in order to study how it would perform. The result was a much more stable and responsive system, as well as a much simpler one as far as hardware. Moreover, we were pleasantly surprised at how easy it was to implement the required software using the Propeller's native language (Spin) which was readily learnable and powerful, as well as a pleasure to use.

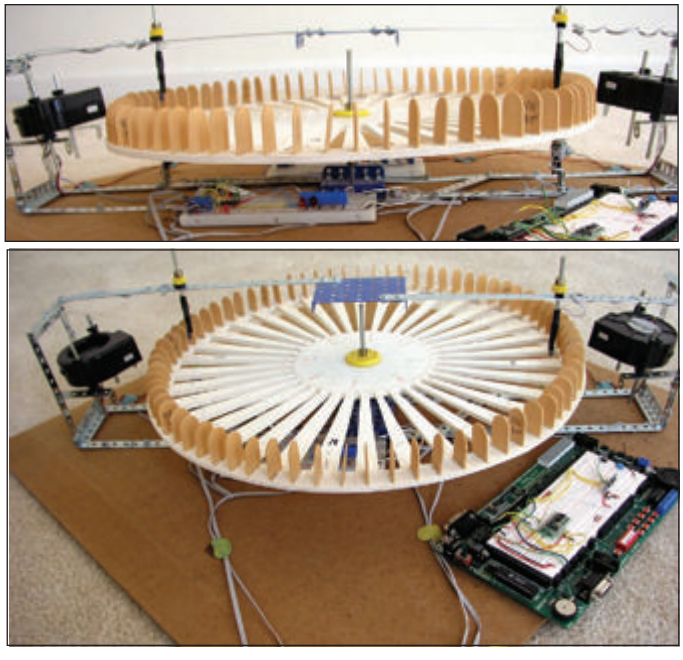


FIGURE 1. Space station model.

The Need For Parallel Processing

Using any of the many microcontrollers available on the market these days (e.g., BS2, SX, PIC etc.), one can accomplish projects that two decades ago would have been beyond the skill level and budgets of the average electronics enthusiast. However, in most projects (e.g., robotics), one has to also use numerous (what we call) "*helper modules*" that take care of tasks that would otherwise require the prohibitive continuous attention of the microcontroller.

For example, robotics projects often require a pulse-width modulation (PWM) system for efficient speed control of DC motors, or for positional control of servo motors. Stepper motors require the microcontroller to continuously change the excitation of the coils to drive the motor from one step to the other. Infrared and ultrasonic distance measuring systems also require the controller to stop other tasks in order to make accurate measurements.

To relieve the microcontroller from having to spend an inordinate amount of time taking care of repetitive tasks, it has become common practice to use various helper modules. These helper modules in themselves are often microcontrollers fully dedicated to taking care of the work for which they were designed.

This, in effect, achieves a parallel processing, multi-tasking system with a microcontroller like the BS2 becoming an overall manager and coordinator of the system. Searching the web (e.g., **www.Parallax.com**), you will find such helper modules as the BiStep motor controller, HB-25 motor controller, PWMPAL, ServoPAL, and PING))) ultrasonic distance sensor to mention just a few.

Most of these modules carry out the continuous

repetitive actions required while the microcontroller communicates with them to obtain data or effect actions using a serial protocol like I²C, SPI, 1-Wire, Asynchronous, and so forth. In addition, these modules encapsulate circuitry that you would otherwise have to create.

If your project requires several helper modules, however, they can add a significant expense to the overall cost of the project. Additionally, wiring and board real estate can become quite cramped. Also, serial communications can be a performance bottleneck, limiting the overall speed of the system. Another disadvantage is that the programmer has to learn many different specifications.

Traditional Solutions

Many microcontrollers (e.g., SX28AC/DP, 68HC12) can implement pseudo-parallel processing using interrupts. This — besides being very complex to program — is not *real* parallel processing. Using interrupts, you can only carry out a small number of tasks before you run out of timing bandwidth. For example, using a buffered software UART while also performing PWM in the interrupt loop can rapidly task most microcontrollers, leaving very little time to do other actions in the main loop of the program.

Another method of designing a parallel concurrent system is to use multiple processors. However, you do need to decide on how these processors will intercommunicate. Usually, this is achieved by designating a central master controller that communicates with the others using serial communication (as in our space station project). However, serial communication can limit the overall system's response time. A much faster and very viable method for intercommunicating multiple processors is achieved by utilizing a shared RAM. However, programming such a system can be overly complex. Imagine what would happen if two processors try to write to the same memory location at the same time, or consider what would become of the validity of the data read by one processor while another is writing to it.

The Propeller Chip Advantage

The Propeller chip has eight processors (cogs) that can run independently and in parallel. With the Propeller chip, you can implement the actions of many helper modules within the one chip. This greatly reduces cost, wiring, and project board real estate. Additionally, with the ability of the cogs to share the 32 KB of onboard RAM memory (called hub RAM), intercommunicating the eight sub-microcontrollers is effortless. The Propeller chip solves all the memory access contentions associated with the shared RAM solution entirely within the chip's hardware. It is so simple to implement concurrent processing with the Propeller that even the novice programmer can achieve astounding results.

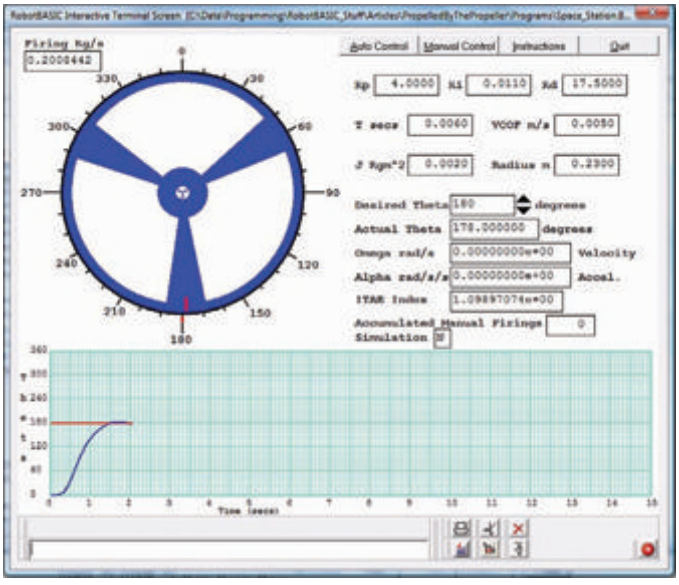


FIGURE 2. RobotBASIC control program.

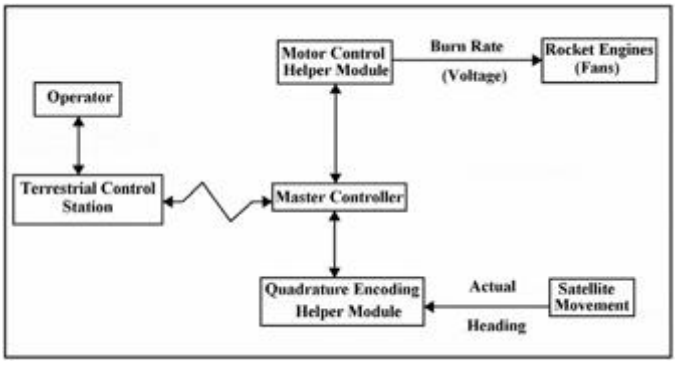


FIGURE 3. The space station model control systems.

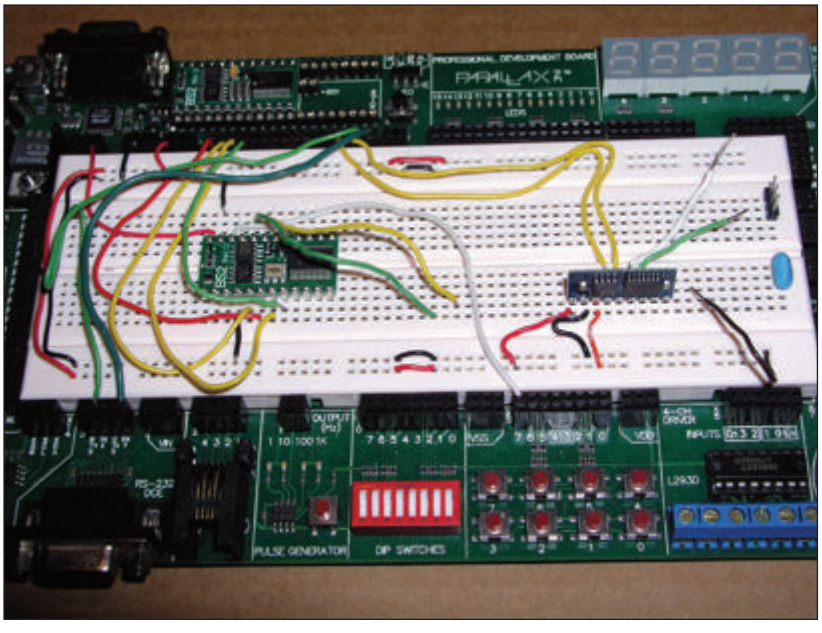


FIGURE 4. BS2-based setup.

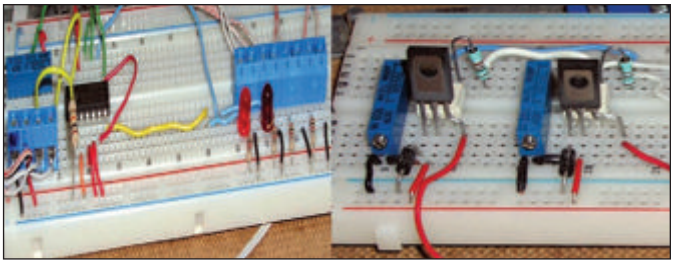


FIGURE 5. Additional circuitry.

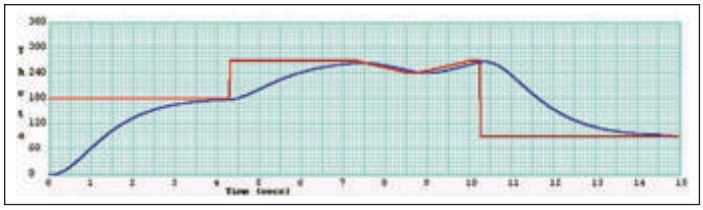


FIGURE 6. Ideal (simulation) step response.

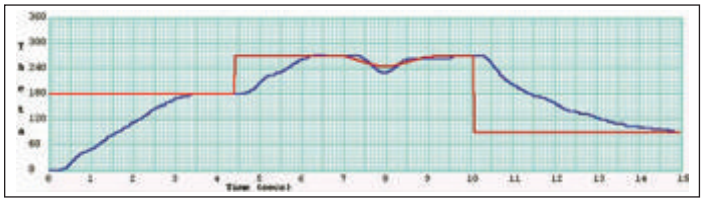


FIGURE 7. Original physical model step response.

You can implement PWM on one cog, a wireless UART on another, a quadrature encoder on a third, an LCD display driver on a fourth, an ultrasonic distance measuring system on a fifth, a PS/2 keyboard interface on a sixth, and so on and so forth. All of these subsystems will be as responsive as if they were each controlled by a fully dedicated microcontroller (which, in fact, they are) and they all can intercommunicate with each other instantaneously through the hub RAM without the associated complexity and time constraints of serial communications or the programming worries associated with traditional shared RAM systems.

Another advantage of the Propeller is called Spin. This high-level language is as easy to use and learn as BASIC, and yet has the advantages of being object-based, giving the programmer much of the power of object-oriented programming. Spin makes it easy to implement fast and powerful actions, and with the object model you do not have to reinvent the wheel for every project. You can use objects written by more experienced programmers to simplify and optimize your programs.

Many clever and dedicated programmers have contributed to a library of objects that accomplish many of the tasks a programmer is likely to need. These objects have been well tested, and are very efficient and fast (usually written in PASM — the Propeller's assembly language). Plus, they can be utilized by even a novice programmer to create professional-level programs. (You can find free prewritten objects in the Propeller Tool software's Library folders at **www.parallax.com/propeller**, and in the Propeller Object Exchange at **http://obex.parallax.com**.)

The Propeller In Action

To illustrate how the Propeller can be used to greatly simplify and boost a fairly complex project we will describe how it helped revamp our space station modeling project.

In the project, we use a PC running a RobotBASIC program (**Figure 2**) that implements the human interface and PID control algorithm to serve as the overall controller. It also allows the user to observe an animated representation of the space station providing visual feedback of the orientation of the model, as well as showing a graphical representation of the historical position over time. In addition to providing automatic control of the model, the system allows the user to exercise manual attitude control by using the keyboard or mouse. The system also acts as a simulator allowing the operator to experiment with the process without the need for the physical model.

The system (**Figure 3**) had a master controller to handle the communication between the model and the PC, as well as the coordination of two subsystems. These subsystems were a quadrature encoder for measuring the heading of the wheel and a PWM system to control the

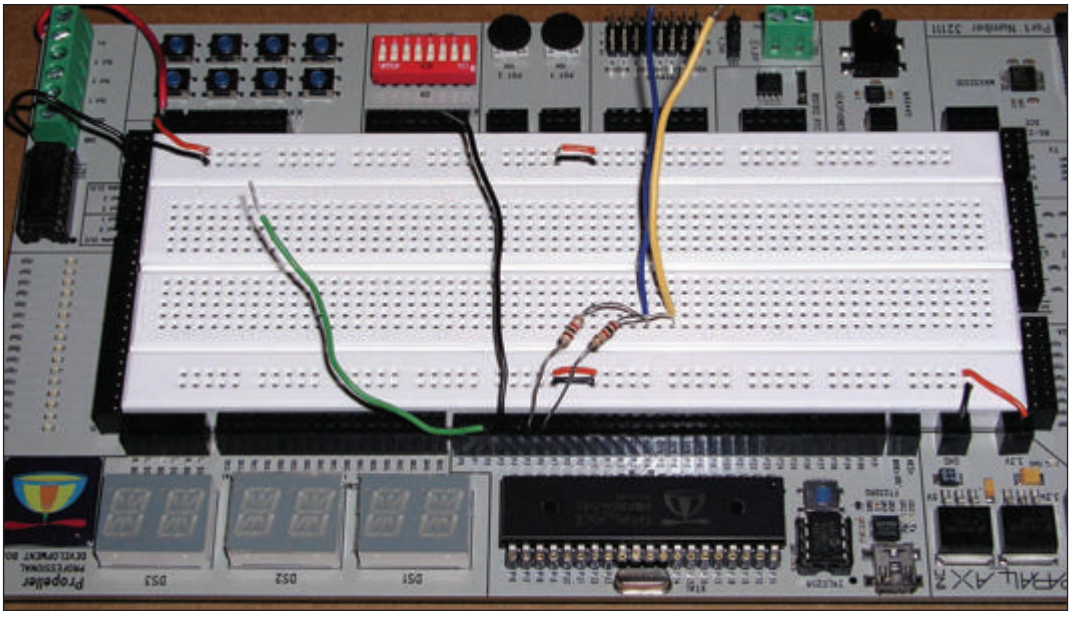


FIGURE 8. The Propeller chip replaced all three controllers in the original setup.

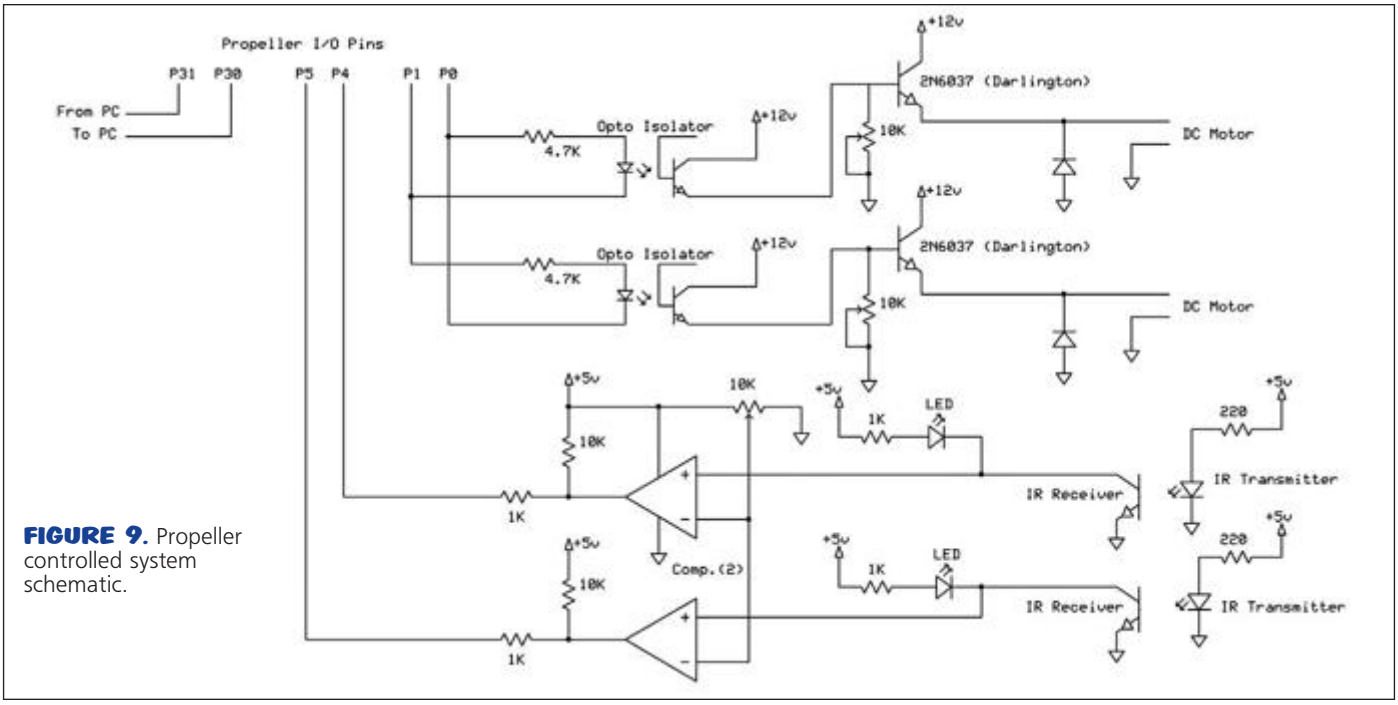


FIGURE 9. Propeller controlled system schematic.

speed of two DC fans for rotating the wheel in either direction. Due to the popularity and ease of programming of the ubiquitous BASIC Stamp microcontroller (BS2), we originally implemented the project using it. The project required two BS2s: one for the master controller and one for the quadrature encoder. Also, a Pololu motor controller was used to take care of the PWM.

The setup of the controllers is shown in **Figure 4**. Notice the two Stamp modules (left) and the Pololu module (right). The board is Parallax's Professional Development Board — a versatile development and experimentation platform with a handy additional DB9 RS-232 connector for easy communication with a PC. **Figure 5** shows some additional circuitry common to both the original and the revamped (with the Propeller) projects. The infrared encoding circuitry is on the left. On the right are the power opto-isolation and drivers for the motors.

The original system performed quite adequately and achieved the project's requirements. Nevertheless, it was not as responsive as we desired and needed to be more robust. You can observe this from **Figures 6** and **7** which show the step responses of the ideal and real systems (i.e., responses [blue] to commanded changes in direction [red]). Due to the limitations of the setup used, the speed of the response of the physical system was not overly fast and not very smooth. The wheel was limited to 2.8 rotations per second (Hz) due to performance limitations imposed by the communications bottleneck between the various subsystems.

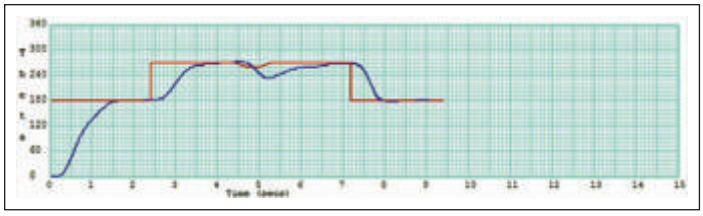


FIGURE 10. Propeller-based physical model step response.

```
{{
  Space_Station_Controller.Spin

-This is the main program for the overall controller.
-It initializes the three objects:
    UART,PWM system and Quadrature encoder
-It sits in a loop waiting for a command from RB
 to set the PWM duty or to reset the quadrature count.
-When it receives the command it acts upon it and also
 sends back to RobotBASIC the value of the quadrature
 count as a two's complement 32 bit integer value.
 The value is in the Little Endian format and RB
 expects that format too.
}}
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000
  PWM_Pin_L = 0
  PWM_Pin_R = 1
  LQuadPin  = 4
  RQuadPin  = 5

OBJ
  D    : "fullDuplexSerialPlus"
  Q    : "QuadratureCounter"
  P    : "PWM_Object"

PUB Main|x,i
  P.Start(PWM_Pin_L,PWM_Pin_R)          'start the PWM cog
  Q.Start(LQuadPin,RQuadPin)            'start the quadrature encoder cog
  D.Start(31,30,0,115200)               'start the UART cog
  repeat
    x := D.GetDec                       'receive PWM duty value or reset command
    if x == -5000                       'code for resetting quadrature count
      Q.Reset                           'if so reset the quadrature count
    else                                'otherwise
      P.Set_Duty(x)                     'set PWM duty value
      x := Q.Get_Count                  'read quadrature count
      repeat i from 0 to 3              'send it as two'complement 32 bit number
        D.Tx(byte[@x][i])
```

FIGURE 11. Main program.

```
{{
  QuadratureCounter.Spin

-You must use the Start method to initialize the object
 and cause the cog to start measuring the quadrature
 count.
-The object will read the High/Low signals on the two
 specified pins and will calculate the number of states
 that have been counted.
-The count can be positive of negative depending on
 counter or clock wise rotations.
-The count is the number of states that have passed.
 Not an angle. To use the count as an angle you must
 multiply by the step size and take into account the
 direction.
-Call the Get_Count method from within the top level
 object to get the state count. Use the Reset method to
 reset the count back to zero.
-Another method that may become needed is Get_Error to
 see if there has been any quadrature miss-steps.
}}
Var
    Byte Error,OldState,CurrState,CogN
    Byte LIR,RIR
    Long Counter,Stack[8]

Pub Start(LPin,RPin)
  LIR := LPin
  RIR := RPin
  Stop
  Reset
  CogN := Result :=CogNew(Quad,@Stack)+1   'start the cog running

Pub Stop
  If CogN
    CogStop(CogN~ -1)

Pub Reset          'resets the quadrature counter to zero state
    OldState[0]     := ina[RIR] + (ina[LIR] << 1)
    Counter := Error :=0

Pub Get_Count    'obtain the state count
  Result := Counter

Pub Get_Error  'indicates if there has been a state error
  Result := Error

Pri Quad
    Dira[LIR] := Dira[RIR] := 0
    repeat
      CurrState[0] := ina[RIR] + (ina[LIR] <<1)   'read the IR states
      if OldState <> CurrState                    'if state is changed
        case OldState                             'figure out if left or
          0: case CurrState                       'right movement and update
               2:Counter--                        'the count.
               1:Counter++                        'if there is a state error
               other:Error++                      'update the error variable
          1: case CurrState
               0:Counter--
               3:Counter++
               other:Error++
          2: case CurrState
               3:Counter--
               0:Counter++
               other:Error++
          3: case CurrState
               1:Counter--
               2:Counter++
               other:Error++
        OldState := CurrState
```

FIGURE 12. Quadrature encoder program.

The Propeller Controlled Model

In the revamped project, the Propeller chip replaced all three controllers of the original system. The schematic is shown in **Figure 9** while **Figure 8** shows the hardware configuration. Other than two resistors, there is nothing there; just four connections to the chip from the hardware shown in **Figure 5**.

The board is Parallax's Propeller Professional Development Board which again is very versatile for experimentation and has the additional DB9 RS-232 connector for easy connection to a PC, but our project used the USB programming port to do the serial communication with RobotBASIC.

The new system is significantly simpler and less costly. It is also much more responsive and robust as demonstrated by **Figure 10** which shows the new system's step response (compare to **Figure 7**). Moreover, the Propeller based model ran quite nicely at 10 Hz and could have accommodated even faster rotations. Due to the Propeller's advantage, we were able to make the sampling rate 10 times faster than in the original system and the overall system's time constant (t) was three times smaller.

Spinning The Programs

The Spin programs are shown in **Figures 11** to **13**. They are well documented and should be self-explanatory. It is important to realize that these programs are not any more complicated than the original PBASIC ones. You can visit **www.RobotBASIC.com** to download the PBASIC and Spin programs, as well as the RobotBASIC program for the simulator and GUI system. Included in the zip file is a PDF document which discusses the details of the PID control and mathematics of the simulation.

Other PDF documents on our website show how to communicate with the Propeller chip from a PC. Programming the new system was actually slightly simpler because Spin performs 32-bit math. This simplified handling of the quadrature count and PWM duty settings. Serial communication with RobotBASIC was simpler too because values could be transmitted as two's complement without the need to do conversion and byte manipulation.

The main program uses three objects. The first two — PWM and quadrature objects — were designed by us in Spin. The third object — a UART for serial communication with the PC — was written by expert engineers at Parallax and is a complex object written in a combination of Spin and PASM.

The availability and ease of use of such *software helper modules* illustrates the power of the Propeller chip and Spin. We used the UART object with ease without having to develop it ourselves and it performed exactly as we needed. The object gave us the ability to do hardware actions (asynchronous serial), and due to the Propeller's multi-processing aspect and the object model of Spin, we were able to implement the actions of a hardware helper module in software without any wiring or expense.

As you study the programs, you may wonder where the parallel processing occurs. Notice that in **Figures 12** and **13** there are the private methods (subroutines) **Quad** and

PWM(), respectively. Examining these methods, you would notice they each have an infinite loop that would never finish. **Quad** continuously monitors and updates the quadrature state count, and **PWM()** ensures the PWM signal is constantly updated.

In traditional interrupt or event-driven programming, these loops would have to be "interrupted" occasionally to allow for the rest of the program to execute. Also, since they would be running together you wouldn't be able to put them in two separate loops like we have them now. Furthermore, due to the interruptions their actions would not be fully *deterministic*, meaning that the system may have signaling jitter and missed pulse counts while other tasks are being handled by the processor.

In the Propeller chip, these two methods run fully in parallel in their own cogs eliminating the need for interrupts. Notice the statements that call **CogNew()** in the **Start()** methods in both figures. This function launches the associated method (subroutine) into one of the eight cogs.

Once a cog is started, it will continuously do its tasks independently of other cogs. However, notice how in **Figure 11** the PWM duty level is set. This is the way to communicate between the cog that is running the main program and the cog that carries out the PWM work. We just set a value within the hub RAM which **PWM()** knows about and can read. Upon reading this value, it will change the duty level of the PWM signal.

Likewise, the quadrature encoder cog will continue to interrogate the infrared hardware and update the state count into a variable in the hub RAM. The cog running the main program reads this count when needed from that variable.

We do not have to worry about contention handling because the Propeller chip does that for us by ensuring that each cog can only read or write to the hub RAM in a round robin fashion. This guarantees that no two cogs can simultaneously read and write to the shared memory.

The round robin contention prevention works on a long (32 bits) by long basis. Should you need to ensure exclusive access to a block of memory over multiple round robin turns, the Propeller chip also provides hardware-based semaphores (eight of them, called locks) which simplify semaphore programming tremendously. In our programs, we do not need them since we are only using longs for the PWM duty and the quadrature state count.

Putting A Fresh Spin On Things

The RobotBASIC program was necessary in the original project because we needed a GUI for the operator of the system to be able to command it. Additionally, the simulator aspect was powerful for the purposes of operator training and for tweaking system parameters like PID factors. With the Propeller-based project, you are able to dispense with the PC system (with a caveat).

With the Propeller chip and a few pre-written powerful objects and some resistors, you can add a PS/2 mouse and keyboard, as well as a VGA (or TV) display to the system. The Propeller can also do floating point math (using an object) which would enable the implementation of an accurate PID control algorithm. You can therefore implement the user interface for the operator, as well as the PID control using just the Propeller chip.

This is definitely a powerful option, but the graphics and GUI are not going to be as powerful as you can achieve on a PC. Also, a PC system can provide additional processing and other services that can enhance a Propeller-based project (e.g., the simulator in this project). **SV**

www.servomagazine.com/index.php?/magazine/article/december2010_Blankenship

```
{{
  PWM_Object.Spin
-You must use the Start method to initialize the object
 and cause the cog to put out the PWM signals.
-The object will use two counters to output a PWM
 signal on the specified two pins.
-To specify the Duty level of the PWM use the Set_Duty
 method. The duty value must range from -1000 to 1000
-Negative values will cause the left pin to be used
 for the PWM level while the right pin is kept low.
 Positive values do the converse.
}}
Var
    Byte CogN
    Long Stack[20]
    Long Period,Duty_L,Duty_R

pub Start(LPin,RPin)
  Stop
  Period := clkfreq/100
  Duty_R := Duty_L := 0
  'start the cog
  CogN := Result := cognew(PWM(@Period,@Duty_L,@Duty_R,LPin,RPin),@Stack)

Pub Stop
  if CogN
    CogStop(CogN~ -1)

Pub Set_Duty(x)
    x := -1000 #> x <# 1000     'limit the value to -1000 to 1000
    if x == 0
      Duty_L := Duty_R := 0
    elseif x < 0
      Duty_L := 0
      Duty_R := clkfreq/100_000*(||x)
    else
      Duty_R := 0
      Duty_L := clkfreq/100_000*x

Pri PWM(A_Period,A_Duty_L,A_Duty_R,Pin_L,Pin_R)|T
  ctra[30..26] := ctrb[30..26] := %00100     'start both counters in NCO mode
  ctra[5..0]   := Pin_R                      'set the pins
  ctrb[5..0]   := Pin_L
  frqa := frqb := 1
  dira[Pin_R]  := dira[Pin_L] := 1
  T := cnt
  repeat
    phsa := -Long[A_Duty_R]                  'update the phase registers for duty
    phsb := -Long[A_Duty_L]                  'and wait for the period
    T += Long[A_Period]
    waitcnt(T)
```

FIGURE 13. PWM controller program.

Then and NOW

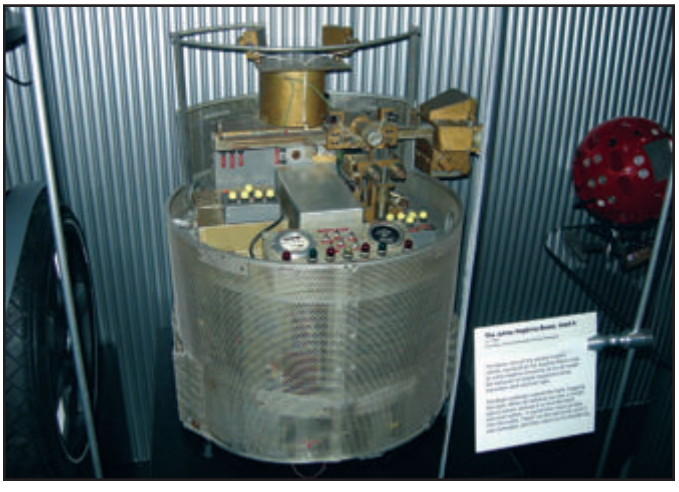
ROBOTS: FROM INDUSTRIAL TO SOME AMAZING CAPABILITIES

by Tom Carroll

*Besides some independently-built creative robots in university labs that exhibited 'animalistic' properties, most early robot products were of the industrial variety. Grey Walter's tortoise robot, Elsie, from the late 1940s was a unique robot for its day and is still considered to be a milestone in the evolution of experimental robotics. The development of the amazing Johns Hopkins Beast built at Johns Hopkins University in the 1960s (shown in a museum display in **Figure 1**) was another milestone in the simulation of basic machine intelligence as applied to a mobile robot base. **Figure 2** shows the Hopkins Beast Automaton II along side Automaton I. These robots could search for a standard black 115V wall receptacle and plug in for 'dinner' when their batteries needed charging — much the same as one of iRobot's Roombas, today. Other university-built robots such as Stanford's 'Shakey' were furthering that new art and science of robotics.*

Despite the exceptional experimental qualities of these robots, the real robots that most people used to associate with the word robot were the first industrial automatons such as the Unimation Unimate that worked in factories. These large machines worked tirelessly in industry, painting, and welding cars. Sparks flew from the welding heads and the media could not get enough photos of this new wave of industrial productivity. Times have changed, indeed. War robots, medical robots, strange crawling robots, tiny robots, and giant robots are headlined in the media and science shows on TV. Quite often, there are a bunch of new robots making the headlines that fit within one category, but I find it interesting to see just how many types of robots in *all* categories are being developed around the world that encompass this emerging technology.

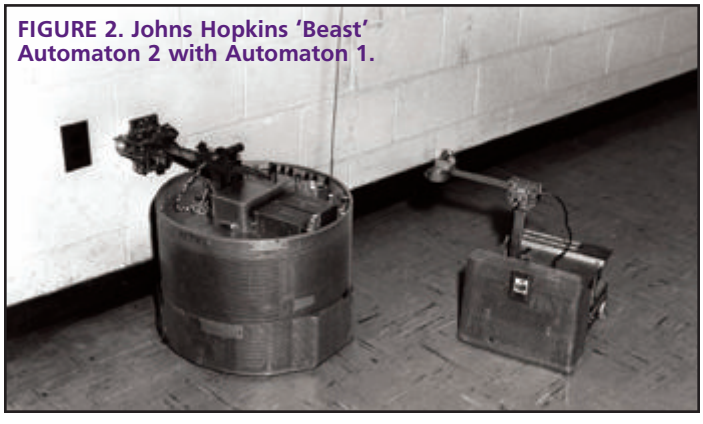
FIGURE 1. Johns Hopkins Beast Mod II.

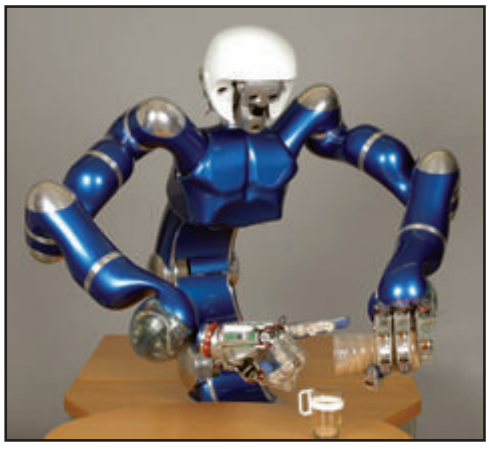


Justin, Germany's New Astro-Bot

NASA is not the only space agency with a robot astronaut. The dexterous humanoid robot, Justin (shown in **Figure 3**), was designed by the Institute of Robotics and

FIGURE 2. Johns Hopkins 'Beast' Automaton 2 with Automaton 1.





FIGURES 3A-B. German Aerospace Center's Justin.

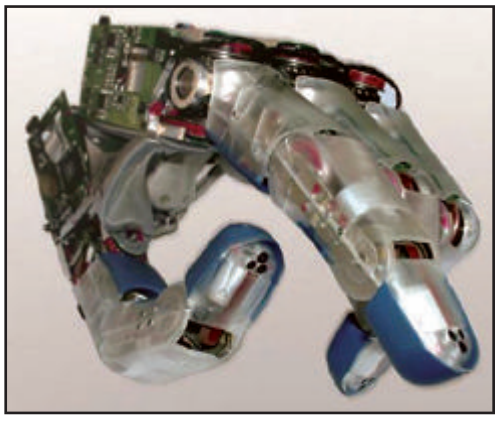


FIGURE 5. Justin's robotic hand.

Mechatronics, part of the German Aerospace Center (DLR) in Wessling, Germany. There are two versions of Justin: a wheeled version to provide a test platform within the terrestrial laboratory environment and another with just a torso, head, and arms to be used in space. The wheeled version shown in **Figure 4** has a unique extendable wheel system that might be applicable to other mobile robots that need stability when dealing with heavy loads.

There are similarities to NASA's Robonaut and Robonaut 2, yet differences. Robonaut is intended for space station servicing missions but Justin will be used to take dead satellites out of orbit or service them on the site. Justin's design purpose is to repair or refuel satellites that need to be serviced, extending the life of on-orbit satellites. Space is full of junk and dead satellites, and ESA feels it might be quite useful to clear out this debris. The robot could be mounted on a dedicated satellite with the express purpose of grasping and removing dead satellites.

The robot has amazing dexterity, approaching that of a human astronaut. Justin's upper body has 43 controllable degrees of freedom with seven in each of the arms, 12 in each of the hands (see **Figure 5**), and five in the torso. The developers chose to use the aRD-concept (agile Robot Development) software architecture in order to avoid the use of the usual predominately monolithic control structure. Instead, they broke it into individual modules which can be distributed on multiple processors. The implementation consists of a small collection of libraries and configuration tools. It allows the integration of a variety of standard tools such as Matlab/Simulink for controller design.

At present, it is used as a teleoperator but the long-term goal is for partial or full autonomy. This teleoperation can be accomplished from a space-borne control panel, much like the space station's Canadarm or the Shuttle RMS arm, or from an earthly location. Control is performed by a head-mounted stereo display helmet for visual feedback from the robot's stereo cameras and arm control is handled by an exoskeleton appendage attached to the controller's arms to mimic their movements. The exoskeleton sends back torque forces proportional to what the robot's arms feel in space. Future control is hoped to be autonomous via verbal commands or simple button commands.

iCub Toddler Robot is a Junior Archer

One robot that I've found to be topping headlines in the US and other countries is the 41 inch tall robot designed by Petar Kormushev and his team of Sylvain Calinon, Ryo Saegusa, and Giorgio Metta at the Italian Institute of Technology in Genoa, Italy. iCub has an amazing 53 degrees of freedom in its arms, hands, legs,

FIGURES 4A-B. Justin's extendable mobile base.

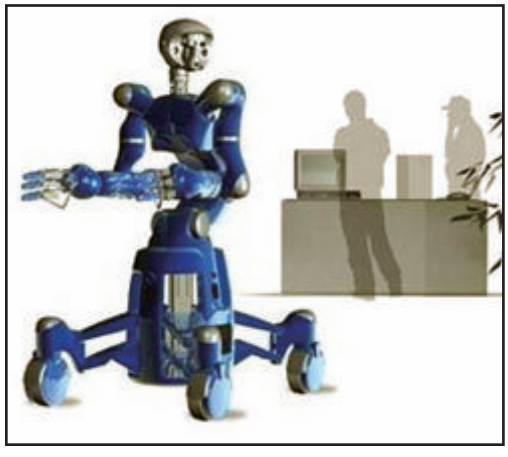


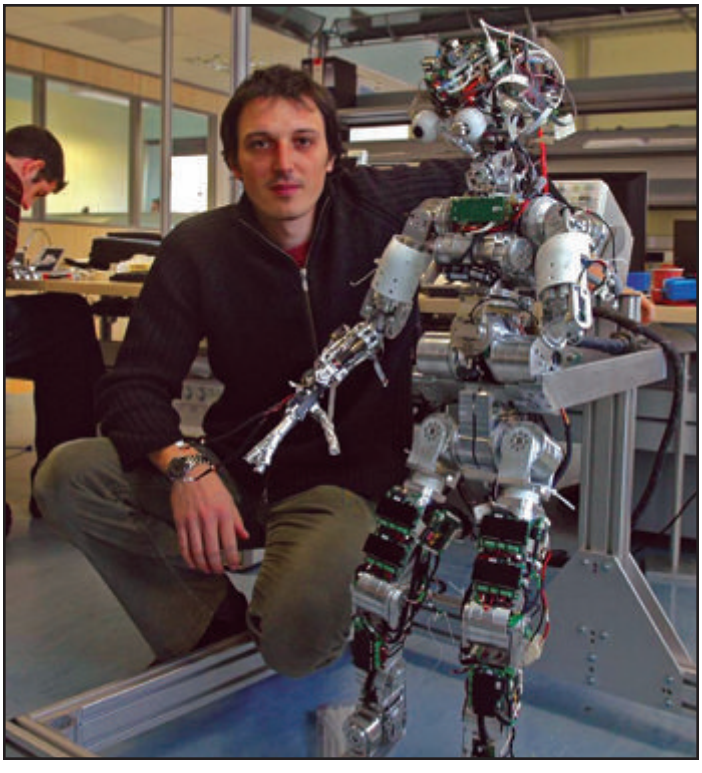
FIGURE 6. The iCub archer robot.

FIGURE 7. The iCub robot misses a few shots at the start.

head, and torso — more so than even Justin. It was modeled after a 3-1/2 year old child. **Figure 6** shows the diminutive humanoid iCub robot topped with an Indian headdress, holding a child's bow with a rubber sucker-tipped arrow, attempting to hit a target a bit over 11 feet away. **Figure 7** might be an indication of iCub's initial accuracy, but we'll give the little guy a break. Now, this might seem a bit ridiculous for a demonstration of a sophisticated robot, but the visual and mechanical skills required in archery utilize most of the capabilities of such a robot.

No, iCub will never compete in the archery events in the Olympics or even become a competent bow hunter, but the open-sourced robotic platform provides an excellent means for proving the chained vector regression algorithm called ARCHER. The title came from Augmented Reward Chained Regression (yes, that's pushing it a bit) and was developed especially for problems with specific knowledge about the goal to be achieved and which have a "smooth solution space," according to Kormushev. For this demonstration, the robot was taught how to hold the bow (note the enlarged rubber grip) and how to pull back the bowstring and shoot the arrow. It learned on its own how to improve its aim and managed to hit the bull's eye after eight tries.

FIGURE 8. Lorenzo Natale and iCub at IIT.



One joint in the index finger was modified to pull back the string and release it after verification of visual sighting, bow location, and positioning of both the left arm holding the bow and the right arm pulling the string. Coordinated movements of the arms are controlled by an inverse kinematics controller. The robot's camera 'eyes' take a picture of the arrows on the target, and an image recognition system determines where the sucker tips of the arrows hit the target. **Figure 8** shows a bit of the complex structure of the robot and its 53 DOF mechanisms standing beside one of the IIT staff, Natale Lorenzo. The Gaussian Mixture Models software filters the colored images of the overall picture and determines what is the target and what is the arrow tip. This visual information is the feedback for the ARCHER algorithm. iCub is also proficient at flipping pancakes.

As adults and even children, we look at such exercises as fairly simple, but the iCub was not developed as an archer but rather as a platform for the development of vision/mechanical control optimization software. For the archery exercise, the ARCHER algorithm is used to modulate and coordinate the motion of the two hands for grasping the bow and releasing the string, while an inverse kinematics controller is used for the motion of the arms. After each shot, the visual information corrects the mechanical positioning of these joints and a new shot is performed. This research will be presented at the 2010 IEEE-RAS International Conference on Humanoid Robots in Nashville, TN on December 6-8. The conference theme is "Intelligent and Dependable Humanoid Robots," reflecting growing interests in developing intelligent and dependable

humanoid robots that can interact with humans to improve the quality of life.

Panasonic Develops a Unique Bed/Wheelchair for the Elderly

I've covered many types of service robots that are intended to care for people, either as a mobile robot within a home or hospital, or as a robotic device that is directly in contact with a human, such as the da Vinci Surgical System or the similar Robodoc by Curexo Technology. Robotic devices are constantly being developed to assist and care for our elderly population. I've spent many years writing about a mobile personal assistant robot that I've designed to act as an in-home aide for the elderly. From my many interviews with seniors at assisted-living facilities, one of the most difficult tasks for many elderly people is the simple act of mobility. Most of us who are physically capable find daily tasks such as walking, getting into and out of bed, and sitting in chairs something so simple we give it no thought. When muscle strength begins to fail, these daily acts become painful and sometimes impossible. The elderly are not disabled in the normal sense as all muscle groups function, it's just that these muscles do not have the necessary strength to function adequately in daily life.

Panasonic has gone a different direction with its new bed that separates to form a wheelchair as shown in **Figure 9**. This allows those with injuries or disabilities to get up without assistance. The user remains in the bed while it turns into a wheelchair. The top half of the mattress rises and the lower half lowers, and the motorized base unit automatically slides out from the bed to form a separate motorized wheelchair. The base is autonomous in that it can detect people, obstacles, and other wheelchairs, and maneuver to avoid collisions according to Panasonic, but the controls are similar to many motorized wheelchairs/scooters. When appropriate, the base unit/wheelchair returns to the rest of the bed system, orients itself, and slides inward to form the bed. Much of their design efforts have been in the development of a simple and reliable separation and reuniting system of the two bed segments. The articulated mattress can also bend in several motions to help people to turn over to prevent bedsores.

The earlier version shown in **Figure 10** was basically a two-part bed system whereas the newer version has the movable overhead canopy. Within the canopy is a flat screen TV and security monitor. Controls allow operation of various home appliances, phone, and view of security scenes. Developed in Japan where a growing segment of the population is elderly, this type of system allows aging people to stay mobile and independent.

Autonomous Shampoo Robot

Figure 11 shows a hair-washing/shampooing robot recently displayed at the Home Care & Rehabilitation Exhibition in Tokyo this past fall. Also by Panasonic, this machine is designed to assist caregivers in helping the elderly with limited

FIGURE 9. The Panasonic robotic bed.

mobility in these seeming simple tasks that require careful handling. Allowing a robot to do this mundane task for patients frees health care workers for true medical and health tasks. This system is designed for hospital and care center use rather than a traditional beauty shop.

Panasonic has long focused much of their robot expertise on serving the needs of the growing senior citizen segment of the population. The hair-washing robot uses Panasonic's robot hand technology. With 16 fingers, the robot washes hair and rinses the shampoo bubbles with the dexterity of human fingers. The robot's two arms scan the head three dimensionally as they move and measure, and remember the head shape to apply just the right amount of pressure to each person when shampooing and massaging. Each arm has three motors that independently control swing, press, and massage motions in conjunction with power detection sensors. The robot even remembers each person's head shape and preferred massage course. Company R&D efforts have resulted in these two prototypes, soon to become part of their product line.

Tandy Trower's New Endeavor

Tandy Trower has left Microsoft to form his own robotics company to address the needs of the growing

FIGURE 10. A simpler version of the Panasonic bed.

FIGURE 11. The Panasonic hair washing robot.

population of seniors requiring some sort of assistance to achieve independent living. Many of *SERVO's* readers know of Trower (shown in **Figure 12**) as the founder of Microsoft's robotics group. Bill Gates personally selected him to form this group in 2005. One of the company's first employees, he was instrumental in developing the Microsoft Robotics Development Studio — a complete software system for a wide variety and level of robot experimenters. The first version of the 'Studio' was shipped in December '06, with 2.0 following in '08.

As their literature states: "Microsoft Robotics Development Studio is a Windows-based environment for hobbyist, academic, and commercial developers to create robotics applications for a variety of hardware platforms. The Microsoft Robotics Development Studio includes a lightweight REST-style, service-oriented runtime, a set of visual authoring and simulation tools, as well as tutorials and sample code to help you get started."

Non-programmers can create robot applications using a visual programming environment. The Visual Programming Language enables anyone to create and debug robotics programs very easily. Just drag and drop blocks that represent services, and connect them up. You can even take a collection of connected blocks and reuse them as a single block elsewhere in your program.

FIGURE 13. The Kompai robot from Robosoft.

FIGURE 12. Tandy Trower of Hoaloha Robotics.

Gates encouraged Tandy in this endeavor but some of the other Microsoft executives were not at all interested in his ideas to assist the elderly, so Trower felt it appropriate to start his own company. He resigned from Microsoft in November '09 to pursue this new adventure.

Hoaloha Robotics

Trower's new company, *Hoaloha Robotics,* is based in Seattle and was formed to develop new software and services to speed the development of socially assistive robots. "Hoaloha is Hawaiian for *caring companion"* says Tandy, and he wants to develop robots to empower individuals who require some sort of assistance. This September, Trower announced that an agreement "to work collaboratively with the French company, Robosoft on the design of a socially assistive robot that can enable individuals who require assistive care to address the challenges to daily living that may come as the result of chronic illness, injury, or aging, and empower individuals to live more independently, with dignity, and at more sustainable costs." Robosoft has more than 25 years of robotics experience and is best known for the Kompaï domestic robot shown in **Figure 13**.

Final Thoughts

The economy of the times is still a bit shaky but inventive minds will never let tough times stamp out great ideas. Justin, iCub, and the many new products being developed to assist our growing elderly population are just drops in the bucket that form the torrent of innovative robotic solutions for our modern world. Whether your robots spring forth from garage workshops, university labs, or large corporation's factories, many of you *SERVO* readers are shaping our new world. Keep up the good work. SV

Tom Carroll can be reached at TWCarroll@aol.com.

ROBO-LINKS

Advertiser Index

Now Serving Europe

with Optimized Logistics

- Currency in EURO
- 3 day shipping to 70% of the territory
- Service in French and English
- Competitive shipping rates
- Huge product selection

One Global Door for Manufacturers

VISIT: www.RobotShop.com

Shipping Worldwide

Robotics at your service! ™

The Future of Servo Control is Calling...

ServoCenter™

Features

- USB, RS232, and TTL serial support
- 16 servos, 16 digital I/O, 8 analog inputs
- Built-in SC-BASIC Sequencer with EEPROM
- Sequencer allows stand-alone operation
- 64 scene presets stored in EEPROM
- Presets instantly loaded or cross-faded
- Built-in configurable smoothing algorithm
- Independent, simultaneous speed & position control
- Scaled, percentage, and group movement commands
- Timed movement commands
- Max, min, & startup position settings
- Ultra-precise 0.05425µS jitter-free pulse resolution

$56.99

ServoCenter 4.1 USB

Flexibility

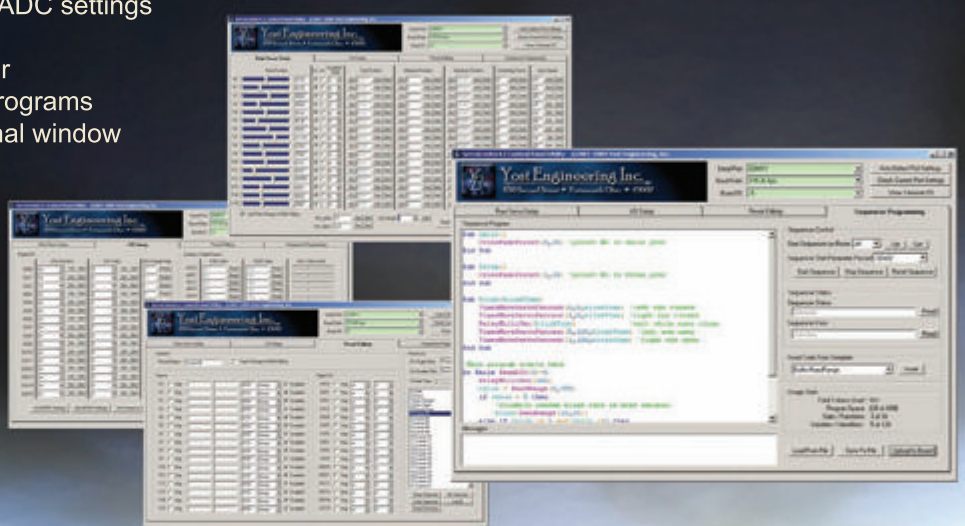
- User upgradeable firmware
- Upload your own firmware with bootloader
- Watchdog timer for failsafe operation
- Over-current, over-temperature, polarity protection
- Internal regulator, external power, or battery power options
- Supports 4.8/6.0V regulated servo supply voltages at 5 Amps
- Each digital I/O and analog input has supply pins
- Direct serial, activeX control, or Win32 DLL communication
- Programming examples in 10+ languages
- Windows, Linux, Mac OSX compatible

$59.99

ServoCenter 4.1
USB Mini

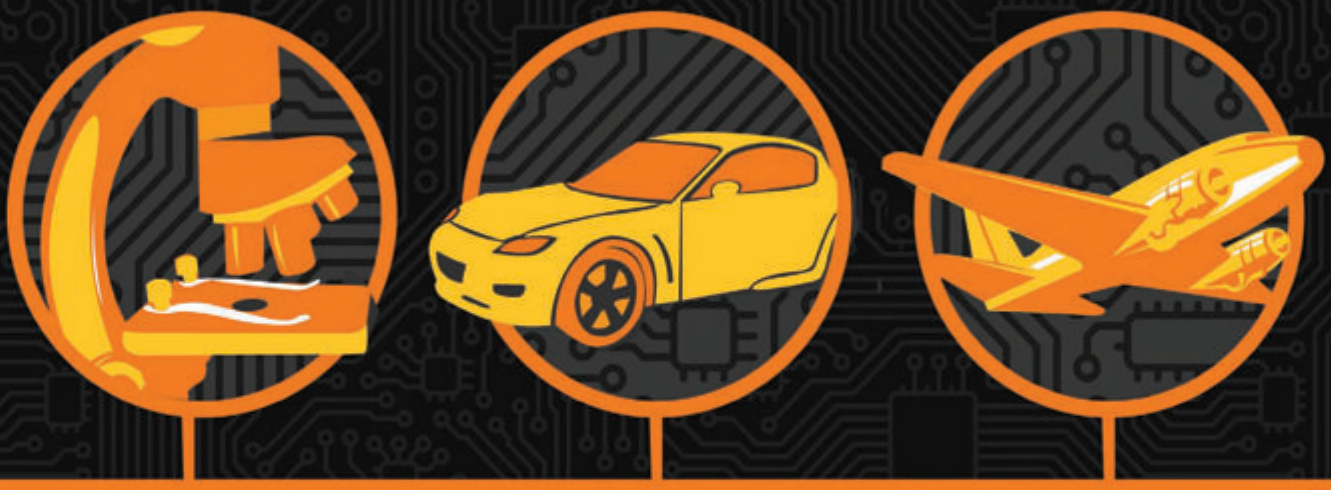
Free Control Panel Software

- Easy editing and configuration of servo settings
- Configure and set up digital I/O & ADC settings
- Edit scene presets
- Program the SC-BASIC sequencer
- Upload and run your SC-BASIC programs
- Debug & communicate with terminal window



www.Servo-Center.com

Yost Engineering Inc.



NO MATTER WHAT THE IDEA

YOUR PCB PROTOTYPES SHOULD BE THE EASY PART

QUOTE & ORDER PCBS ONLINE AT **WWW.SUNSTONE.COM** OR CALL **1-800-228-8198**

SUNSTONE CIRCUITS

THE EASIEST PCB COMPANY TO DO BUSINESS WITH

ValueProto™

PCBexpress®

Full Feature

Sunstone Circuits® pioneered the online ordering of printed circuit boards and is the leading PCB solutions provider with more than 35 years of experience in delivering quality prototypes and engineering software. With this knowledge and experience, Sunstone is dedicated to improving the PCB prototyping process from quote to delivery (Q2D®).

Did You Know? Sunstone Offers:

- Controlled impedance testing
- Fine lines and spacing [.003]
- RoHS compliant finishes
- Free 25-point design review
- Free shipping & no NRE's
- Flex / Rigid Flex Boards
- Online Quote & Order
- PCB123® design software
- RF / Exotic Materials
- Over 99% on-time or early delivery
- Best PCBs in the industry
- Live customer support 24/7/365